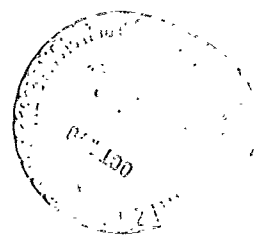
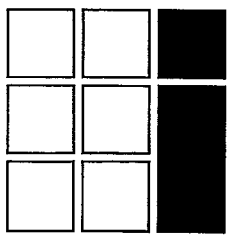




INTERMETRICS

CR 108654

Final Report

Multiprocessor Computer System Study

By

James S. Miller, Daniel J. Lickly,
Alex L. Kosmala, and Joseph A. Saponaro

March 1970

Prepared under Contract NAS 9-9763 by

INTERMETRICS, INC.
380 Green Street
Cambridge, Massachusetts 02139

FOREWORD

This document is the final report of an engineering study of Multiprocessor Computer Systems, and the development of multiprocessor theory and problems related to the proposed space station and space base data management systems. This research was sponsored by the National Aeronautics and Space Administration Manned Spacecraft Center, under contract NAS 9-9763, and performed by Intermetrics, Inc., Cambridge, Massachusetts. Dr. James S. Miller was the Technical Director of the effort.

The study program covered the period from June 27, 1969 through March 27, 1970. Mr. James P. Ledet (EB6) of the Manned Spacecraft Center, was Technical Monitor

Table of Contents

Chapter 1

Study Objectives and Terminology

1.0 Introduction

Multiprocessor and multicomputer systems are being proposed for advanced manned space applications, including the space station and space base programs. Since the applications of systems of this type are relatively recent and not uniformly successful, the Manned Spacecraft Center has undertaken an analysis and extension of this technology. This document contains the results of a study of multiprocessor computer systems, related theory, and potential problems associated with implementation of such a system in a space station Data Management System (DMS).

This report is presented in six chapters. This chapter presents the background and objectives of the study, and a review of terminology involved. Chapter 2 presents the relevant results of a survey of existing multiprocessor flight and ground-based systems. Chapter 3 presents design considerations which are relevant to the architectural configuration of the data management computer system. Chapter 4 discusses the major design guideline constraints and requirements applicable to the system. Chapter 5 presents the architectural design of a multiprocessor computer system capable of satisfying the requirements. Chapter 6 presents recommendations for future work in the areas of technology and design.

1.1 Background

Among the next generation of space vehicles are to be craft which closely resemble earth-based scientific facilities. Earth orbital space stations and bases are being defined as facilities which can support operational and experimental equipment on a long-term basis of about ten years. An earth-orbiting space laboratory housing 12-50 men has been proposed by NASA for implementation in the 1975-1980 period.

The space station and the subsequent space base program will introduce a new, more routine mode of space operations than has characterized past programs. The station will exploit the unique features provided by its location in low earth orbit (200-300 km; inclinations of 28.5° to 55°) for rapid earth viewing and unobstructed celestial viewing, and will allow scientists and engineers to pursue a wide variety of research and application activities on board. It is also currently envisioned that men and equipment will be ferried between these stations and the ground in a reusable Advanced Logistics System (ALS) shuttle vehicle. Obviously, to achieve these goals, many of the functions previously accomplished on the ground will be

performed on board the base, allowing it quite autonomous operation.

Part of the on-board Data Management System will be a computer system capable of supporting all required mission operations. Existing space-qualified information processing equipment lacks the capacity and flexibility to satisfy the diversified space station/base data management requirements.(1) The current inventory consists primarily of dedicated hardware designed for specific applications. Various government and industry programs are developing technology and hardware applicable to the space station data management requirements. Anticipated advancements in electronic technology indicate that much can be done over the near future to improve reliability, weight, power, and volume of flight hardware. Included within this scope are multicomputer and multiprocessor computer systems, which are being proposed as candidates for the space station/base DMS since they seem to offer the reliability, expandability, and modular features required.

1.2 Objectives of the Study

The principal objectives of this study were threefold:

a) Collect and analyze available existing knowledge concerning actual and proposed multiprocessor system design, utilization and communication techniques.

b) Incorporate this knowledge in a generalized multiprocessor theory that establishes a baseline definition of the various system configurations and problems as defined to the present date.

c) Expand multiprocessor computer system theory to include the system and communication problems that would be encountered in the DMS application.

The methodology used to accomplish these objectives began with a survey of the state of art of multiprocessor and multicomputer systems. Twenty-nine ground based and airborne systems were reviewed and information collected as to their current status, architecture and organization, principal features of hardware and software, problem areas, and applications. In conjunction with this task, basic definitions and elementary concepts in computer architecture and memory technologies were documented.

Several basic multiprocessor configurations were analyzed in terms of their applicability to the expected space station requirements, and in particular their reliability and expandability. After a preliminary analysis of these approaches, a parallel effort was established to examine some of the basic design considerations and tradeoffs in multiprocessor systems.

This included an analysis of reliability, memory paging and segmentation, interrupt schemes, display concepts, stack, microprogramming, and storage protection.

To embody the knowledge gained from the survey, extension of multiprocessor design, and projection of hardware capabilities, a system design was prepared. While confined largely to the organizational level, this design satisfies the requirements presently envisioned for the orbiting laboratory program.

The text of this report generally refers to the space station/base application as simply the space station, to eliminate redundancy. However, it is believed that the computer systems for both should differ in scale only, and therefore the material presented herein is intended to be applicable to both.

1.3 Introduction to Computer Architecture

This section is included in this report to define some of the language used in present-day descriptions of computation systems. The vocabulary and usage described are intended to represent a majority opinion from the computer industry. In places where such a majority does not exist or is not very substantial, alternate descriptions are provided. This approach tends to discourage any attempt at rigorous or strict treatment of the subject; indeed, some areas have been left somewhat imprecise. However, it is believed that the basic concepts can be presented clearly and briefly by use of simple language and examples.

1.3.1 Hardware Elements

Although most people have a fair idea of what a computer is, there is no easy way to specify a set of criteria by which one might satisfactorily decide whether a given collection of equipment is or is not a computer. As far as the present discussion is concerned, the only systems meant to be included are those which are computers without doubt. Almost any such system will contain one or more units of each of the following types, plus the communication and data paths required to interconnect them:

a) Memory

b) Processor

c) Input/Output Controller

d) Input/Output Device

Descriptions of each follow.

1.3.1.1 Memory

Often the largest and most expensive element of a computer is its memory. Most computers have a number of different storage media. Usually these are segregated into categories of high-speed units and secondary storage units. The latter, which generally involve mechanically moving parts and are accessible only via I/O instructions, are covered under the I/O Device category.

The remaining memory is almost always of the random-access type; that is, the time required to obtain each word from a given unit is the same. Examples of the kinds of unit which do not have this characteristic are the delay-line memory and the Hughes Dynabit memory; in both, values nearest the output end of the unit are accessible most rapidly, and so on.

The two classes of random-access memory are read-only and read-write. The read-only memories usually have their contents manufactured into them; thus, changing the contents of a read only memory requires physical modification of the device. Such memories are generally less expensive than a read-write unit of the same capacity and speed, and are used in applications where the contents do not need to be, or are not allowed to be, changed. Read-write memories, on the other hand, are designed so that their contents are electrically alterable, although in some instances R/W memories are utilized in a manner which causes them to normally behave as read-only devices.

R/W memories may be classed as volatile or non-volatile according to whether their contents remain intact when power is removed. Core memories are non-volatile, since the medium of storage, the polarity of residual magnetic flux in the core, is self-sustained; flip-flop memories, on the other hand, are volatile, since the state of the device is sustained only by its energization.

Non-volatile memories may be further classified as destructive-read-out (DRO) or non-destructive-read-out (NDRO). Core memories are DRO devices, since the contents of the selected cores are sampled by driving the magnetization state of each to the "zero" condition, which generates an induced voltage in the sense lines of those cores which were in the "one" magnetic state. This action leaves all of the interrogated cores in the "zero" state. Thus, the information originally contained is instantaneously erased from the cores, and must be rewritten if the same contents are to be subsequently accessible. Plated-wire and flat-film memories, however,may be non-destructively read. There are two major advantages for NDRO memories: First, they are less vulnerable to power fluctuations or other interruptions of the read process, since at no time during the reading operation is the information cleared from the memory. Second, they can be made to operate at a higher reading rate, since it is not necessary to follow each read by a rewrite operation.

Many computers use several types of random-access memory. For example, in the IBM 360/65, read-only and volatile read-write memories are used in the processor, while DRO memories of two speeds are available as main storage. The term main storage, which is synonomous with main memory and, unfortunately, processor storage, refers to that memory used for storage of instructions and data of programs being executed. Blocks labeled as memory in diagrams of computer organizations will nearly always refer to main-memory units.

To enhance the speed gains achievable from overlapped computer and I/O operations and from multiprocessing (see below for descriptions of these), memory interleaving is often used. This technique consists of organization of the main memory into independent modules capable of simultaneous operation, and the distribution of memory addresses among them. For example, if memory is divided into eight modules, locations 0, 8, 16, 24, etc. would be physically located in the first module, locations 1, 9, 17, 25, etc. in the second, and so on. Based on the presumption that instructions and I/O transfers largely use sequential memory locations, this method of interleaving tends to decrease the probability of memory-access conflicts.

1.3.1.2 Processor

The processor units in a computer may be roughly described as those units which decode and execute the non-I/O instructions from the programs. I/O instruction execution in many computers takes place in an I/O processor to avoid interference with the main or central processor. Configurations of processors vary widely; in the most straightforward case, the processor is essentially a single entity. At the other extreme, however, processors are comprised of special-purpose execution and decoding elements many of which may be (and hopefully are) operating simultaneously. For example, the CDC 6600 main processor contains a decoding unit and ten special-purpose execution units. Processors may be segmented for other reasons as well. For example, the JPL STAR computer, designed for exceptionally long life without maintenance, has a processor divided into five parts, with spare copies of each, to enhance its reliability; the ILLIAC IV computer separates instruction decoding from execution, and incorporates 256 elements driven by a single decoding unit.

1.3.1.3 I/O Controller

Early computers were built without separate controls for I/O operations; as a consequence, program execution would proceed at processor speed until an I/O instruction was performed. At that point, the processor became dedicated to the performance of the I/O operation. When that operation was completed, normal instruction execution was resumed. The effect of treating I/O

instruction execution like other-instruction execution was to reduce the effective speed of the processor towards that of the I/O devices used.

To eliminate this bottleneck, I/O control hardware was added to the system. In some current computers, it may be in the processor unit, and in others it is a relatively more independent component. In the IBM 360/65, as an example, the equipment is referred to as a channel; channels are placed into operation by execution of a start-instruction in the processor, and thereafter operate by fetching the executing commands themselves concurrently (overlapped) with processor operation. Many channels may be in simultaneous operation, and one type of channel, the multiplexor channel, can control the concurrent operation of a multiplicity of I/O devices. In spite of this high degree of overlap, it is sometimes true of large scale data processing systems that the processor is idle a disappointingly large fraction of the time, waiting for I/O operations.

1.3.1.4 I/O Device

Into the category of I/O devices fall the drum, disk, and tape units, card readers, printers, and the like. In real-time systems, transducers of wide variety are used. Additionally, the I/O mechanism may be used to control the operation of hardware not usually considered to be I/O devices, such as large-core-memory (LCM) and even other computers. The advantage of such operations is that the processor is freed from the speed constraints imposed by these elements in a manner completely analogous to that used with ordinary I/O devices. Again using the IBM 360/65 as an example, the LCM cycle time of 8 μsec is substantially slower than the main storage cycle of 0.76 μsec; use of the so-called storage-channel can therefore increase the overall efficiency of data transfers between memories significantly.

1.3.2 Program Structure

A prerequisite to the understanding of the many configurations of computer systems currently available is an understanding of program structural organization. Although what is to be described is not universally recognized or even always relevant, the trend is in that direction, for reasons which will become clear as the discussion continues.

Again looking back to early computers, a program was simply described as a group of instructions with an identified starting location and one or more ending locations. A program was loaded into the memory of a computer, and execution was begun and allowed to continue non-stop to the end. This simple structure is quite unsatisfactory from an efficiency point of view, and has gradually been abandoned. In its place is a view of a program as a time-

varying group of associated processes or tasks whose constraints on relative execution sequence are dictated by precedence relationships associated with each.

The development of monitors or operating systems (also executives or supervisors) has stimulated the above view of programs. At first, monitors were developed to automatically load and execute jobs sequentially without manual intervention. The conceptual generalization of this system led to execution of more than one job concurrently, in the sense that while one job awaits completion of an I/O operation, another job can be partially executed by the processor which would otherwise be idle. This is an elementary form of multiprogramming; the more general form presently used recognizes the possible separation of single jobs into several tasks or processes which may themselves be executed concurrently in the above sense. A second kind of multiprogramming has been facilitated by the hardware of the Honeywell 800, 1800, and 8200 computers. Referred to by Honeywell as horizontal multiprogramming (as opposed to the vertical multiprogramming described above), this form of processor-sharing takes place on an instruction-by-instruction basis. The processor of the H1800, for example, contains eight groups of the registers used for program control (instruction counters, index registers, etc.), and each set may be used independently to execute programs. Because there is only a single instruction decoding and execution unit, only one instruction is executed at a time, but control is frequently passed from one active group to the next. Groups which are waiting for the I/O are automatically bypassed, so that those groups which can use the CPU are given access without software intervention.

Along with operating system development came the concept of resource allocation which governs the assignment of core space, tape drives, etc., and even units of data filed on secondary storage, to requesting tasks; it was a natural extension of resource allocation to consider processors themselves as resources which could be requested and released by tasks as necessary. Thus, the execution by a task of a pseudo-instruction which specifies that the task cannot continue until a specified event has occurred can be interpreted as the release of the processor by that task; the occurrence of the awaited event then is noted as a request by the waiting task for a processor. It is easy to see how this organization of program lends itself to use in systems which contain more than one processor (multi-processors). However, extension to more than one processor usually introduces two classes of interlock problems whose solution is not necessarily straightforward. The two classes of interlock problems are: first, the incorporation of means to prevent simultaneous operations on a single data base where this is not logically permissible, and second, the prevention of system collapse because of situations such as the one where two or more tasks are stalled, each one of which is waiting for another stalled task to perform some operation. The solution of the first interlock problem has been greatly aided by the addition of a non-

interruptible test-and-then-set instruction to the hardware. This kind of instruction causes the contents of a location in memory to be tested and then altered in a single memory cycle, so that there is no "gap"during which another process, processor, or channel can gain access to the same location. If this location is respected as a "lock" by the software, a process wishing sole access to the protected data executes the test-and-set instruction, which always leaves the lock locked. The process then uses the result of the test to see whether the lock had already been locked; if so, it must wait for the process which locked it to finish its use of the data and unlock the lock.

Units of program which are subject to sharing in a multiprogramming or multiprocessing system must be treated in accordance with their categorization as not-reusable, serially-reusable, or reenterable.

Program units are not-reusable if they modify themselves during execution in such a way that a second attempt at execution will fail. A program which does not modify itself or which re-initializes itself upon subsequent use is called serially reusable if the same copy may be used repeatedly, but by only one process at a time. Reenterable program units are those which may be used concurrently or simultaneously by more than one process.

Another classification of program units specifically refers to whether they modify themselves or not. The term pure procedure means that the program unit (procedure) does not modify itself. Such units may or may not be reenterable, but they are at least serially reusable. Obviously, programs intended for execution from read-only memory must be "pure" in this sense.

1.3.3 System-Use Classification

Computer system use is commonly classified into three categories. Smaller systems are often totally dedicated to a single category, although increasing numbers of larger systems are capable of performing two or three with reasonable success. The first category to exist historically is batch-processing. In current usage, the term refers to a mode of operation in which programmers submit their job decks for computation expecting an interval of hours or longer before their results are returned. In batch-processing, no interaction is possible between the programmer and his computation.

To remove the long delay between submission and computation, time-sharing systems have been developed which allow many users, from remote terminals, to use the computer as though they were each the only user. That is, commands issued from a terminal are executed by the system immediately, and results are displayed almost at once.

In real-time applications, even more immediacy is required, since the computer is typically in a control-loop, and must issue control signals promptly, as a function of the input values it automatically receives from the controlled system.

1.3.4 Configurations

Figures 1.1-1.8 show a number of system configurations in current use. The interconnections shown represent the primary data and control paths, without respect to their mechanization, which is the subject of the next section. Figure 1.1 shows the conventional single processor or uni-processor system; the processor is connected to the memory and to the I/O controller, and the I/O controller is also connected directly to the memory. The connection of the I/O controllers to I/O devices, and the I/O devices themselves, are not shown in the figures, but are understood to be present.

In Figure 1.2, the organization of the CDC 6000 series and the related CDC 7600 is shown. The peripheral processors (PP), significantly smaller and less powerful than the main processor, are used to perform the house-keeping functions involved with job setup and control, and are responsible for execution of I/O operations. The I/O functions normally found in I/O controllers in other computers exist largely in the PP's in this series of systems. The main processor is spared responsibility for the less taxing operations and therefore has a higher availability for execution of the meat of the problem programs. As mentioned previously, the CDC 6600 (and 7600) main processors internally embody some degree of multiprocessing to enhance their speed.

A multicomputer system is shown in Figure 1.3, and represents the IBM 704x/709x Direct-Coupled System (DCS) and the IBM 360 Attached Support Processor (ASP) system. Like the CDC 6000 series, the processors in the system are dissimilar, with the less powerful one used for housekeeping and I/O operations, in an attempt to allow the more powerful one to concentrate on the number-crunching kernels of each job. That this configuration is described as a multicomputer rather than a multiprocessor system stems from the fact that the processors do not share memory; they communicate only via a direct processor-to-processor link and via a channel-to-channel adapter which makes each computer look like an I/O device to the other. The two systems may also share I/O devices, such as disk storage.

The dual-computer system shown in Figure 1.4, the IBM 4 Pi CP-2 configuration for the F-111 Mark II Avionics system, is similar to the DCS or ASP since the processors communicate with each other only directly or through the I/O interface, but do not share memory. The system is different, however, in that the processors are identical, and the hardware and software are

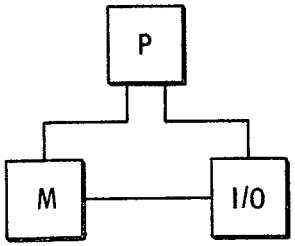


Figure 1.1 Uniprocessor Configuration

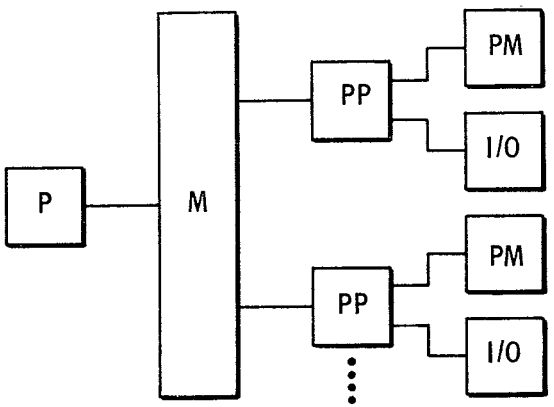


Figure 1.2 CDC 6000 Series

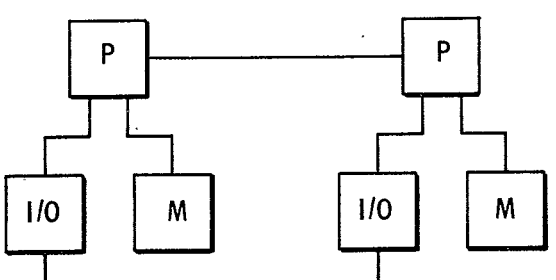


Figure 1.3 IBM DCS & ASP

designed so that the failure of one processor or memory can be tolerated to the extent that the system can continue to perform its major functions. This characteristic has led to the use of the term federated dual-computer system.

Figure 1.5 portrays a modular computer structure investigated by the NASA Electronics Research Center. In this design, each processor has been split into control and arithmetic units for reliability reasons. Not shown in the figure is a configuration assignment unit (CAU) which is capable of changing the connection paths between the other units. Again, in this configuration, no memory may be shared, although more than one memory unit may be connected to a processor. Although these units typically function as independent computers, a potential connection scheme is to add voting logic at appropriate places and to cause all units to execute identical programs. This provides a system of high reliability for critical computations when insufficient time exists to diagnose errors and reconfigure the system. During less critical times, the units might perform independent calculations, or one or two could be idle as standbys in case of failures in active units.

The MIT/IL computer designed for a high reliability application is portrayed in Figure 1.6. Two copies of each module are included in the design, although one processor is always in a standby condition. This sywtem is therefore not really either a multicomputer or multiprocessor system even though it utilizes two processors. Both memories accept all write commands from the active processor, although only the one instantaneously designated "primary" responds to read requests. As a result, both memories should contain identical contents, so that if an error is detected in one, the designation of "primary" can be switched to the other and the read request repeated. The JPL STAR computer is similar, in that its operation is essentially that of a uniprocessor even though multiple copies of modules are present in the system. These two systems have been mentioned to show the existence of "gray areas" of system classification.

The most frequently utilized general architecture for a multiprocessor configuration is shown in Figure 1.7. This configuration is found in the IBM 360/65 multiprocessor, the Burroughs D825, 5500, etc., Univac 1108 and AN/UYK-7, IBM 4 Pi EP for VS A-NEW, IBM 9020, Hughes 4400, ERC EXAM, MIT/IL ACGN computer, and so on. The distinguishing characteristic of the multiprocessor organization is the equal sharing of memory and I/O by each processor. Although the processors in this organization are often alike, they need not be. When they are, the operating system software usually treats them interchangably, and tasks may be assigned to any one when they become ready for further execution.

The ILLIAC IV system shown in Figure 1.8 is called an array processor, and differs markedly from the other systems

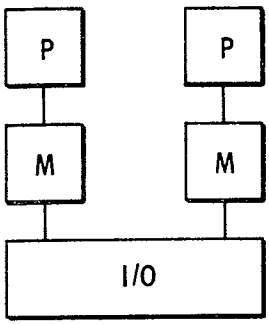


Figure 1.4 IBM Mark II Avionics Computer

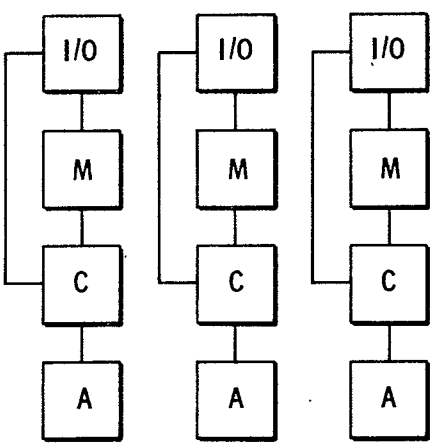


Figure 1.5
NASA/ERC Modular Computer

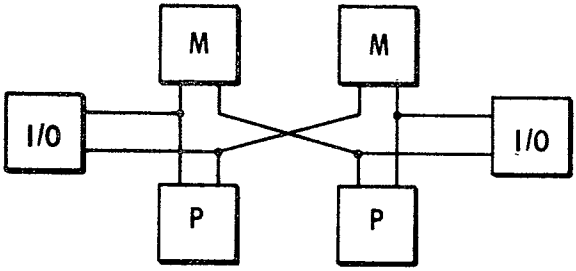


Figure 1.6 MIT/IL SIRU Computer

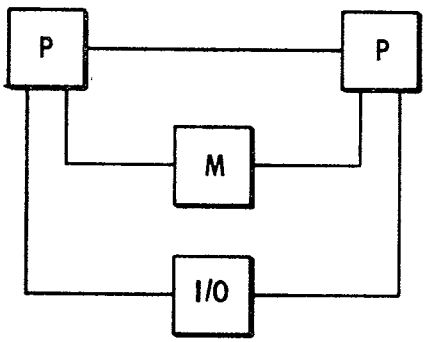


Figure 1.7 General Multiprocessor

described. Associated with each of the four instruction decoding units are 64 execution units, each with its own memory. Thus each instruction can be executed by up to 256 units each addressing data in its own memory. The execution units are arranged in a linear string but interconnected as though they were laid out in four square matrices; that is, processor i can communicate with processor i+1, i-1, i+8, and i-8, with end-around connections where necessary. This system is driven by a Burroughs 6500.

1.3.5 Interconnections

Figures 1.9-1.11 illustrate three interconnection techniques used to tie processors, memories, and I/O controllers together. The scheme which is conceptually simplest is shown in Figure 1.9, and consists of a common data bus to which all modules are attached. This bus, which could be of word, byte, or single-bit width, is time shared between pairs of units which wish to communicate; only one message at a time is possible in the simple system shown. This restriction has advantages as well as disadvantages. The most pronounced disadvantage is the bottleneck imposed by the one-at-a-time communication limitation, since as processors and memories are added to expand the system capacity, waiting times for bus-access grow and reduce the per-unit effectiveness of the system. The advantage of the single bus lies in its conceptual simplicity; conceivably, it could be a single wire with only one connection point per module, although practicality requires considerable logic in each unit. Also, the single bus permits implementation of certain data-interlocking requirements by simple brief monopolizations of the bus by the processor involved, without the need of any explicit software or hardware locking machinery. Finally, the simplicity of the data bus provides the minimum difficulty in adding a unit to the system: it is simply attached to the bus.

Figures 1.10 and 1.11 illustrate two versions of an essentially similar interconnection technique. In the scheme shown in Figure 1.10, units called multiport memories permit multiple connections to each memory module. A conflict-resolving switch in each memory awards access to one requesting-unit at a time, but when several units simultaneously request access to different modules, all of these accesses may be concurrently granted. As in the common data bus scheme, data path width in these busses may be of any convenient size. However, the number of ports on each memory unit is decided when the unit is built, and if all ports are used in a given system, addition of a processor or I/O controller is impossible.

A crossbar style of interconnection is shown in Figure 1.11. This scheme is similar to that of Figure 1.10 in that simultaneous communications are possible, but it differs since the memories are single port devices and the switching is accomplished in an external unit. That this mechanism can grow

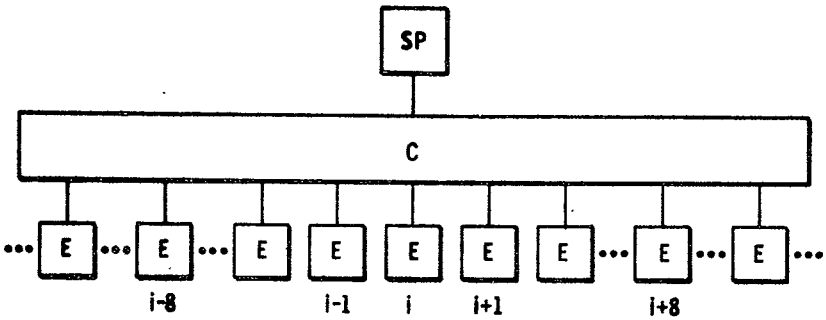


Figure 1.8 ILLIAC IV Computer

(SP = supervisory processor, C = control processor,
E = execution processor)

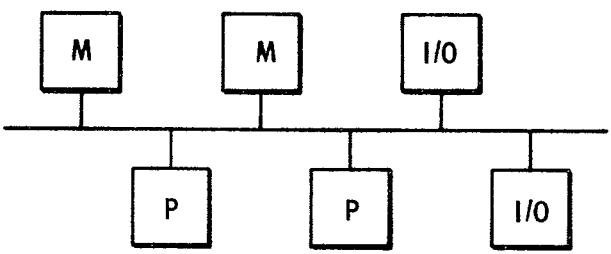


Figure 1.9 Common Data Bus Connection

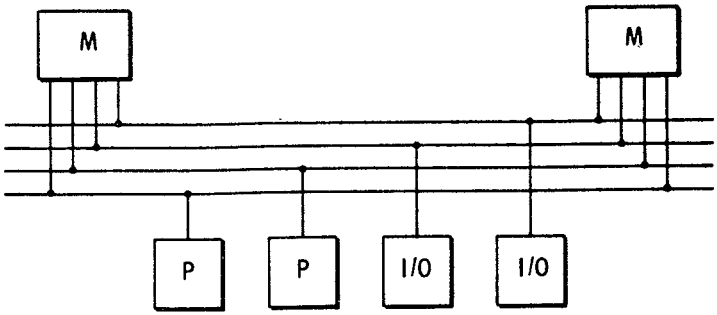


Figure 1.10 Multiport Memory Connection

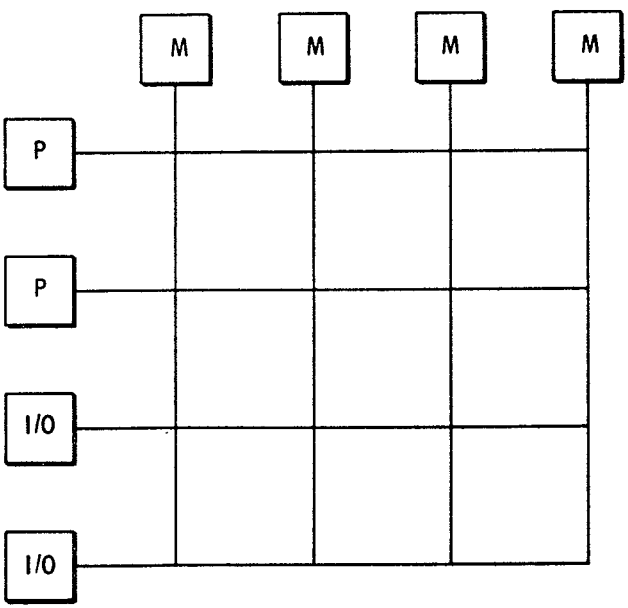


Figure 1.11 Crossbar Connection

to be quite complex is illustrated by the fact that the maximum configuration of the Hughes 4400 multiprocessor switch has a component count equivalent to 2.5 processors. Since the switch is external to the other units, it is possible to design it to be modular and therefore expandable, to avoid the capacity limitation imposed by multiport memories. A modular switch of this kind has been under development at the NASA ERC.

Chapter 2

Survey of Multiprocessor and Multicomputer Systems

2.0 Introduction

This chapter presents the results of a survey of existing and proposed designs for multicomputer and multiprocessor architecture. The survey has included reviews of vendors sales and technical material, articles from technical journals, and discussion by telephone and in person with vendors and academic computer personnel. Material from these sources has been combined with the technical experience of the authors and summarized in the following pages. Although a common general format has been used, the information given, and in fact the information which exists, varies considerably among the summaries. For the most part, this is a result of our attempt to include only information which is fundamental to the individual system or which is believed relevant to considerations of the design of a computer organization suitable to meet the projected requirements of a space station mission.

Although only relevant material is documented in this section, the survey-taking process gathered other information which provides the bulk of survey material that is described elsewhere. Computer design examples are presented in Appendix A, which reviews in detail the segmentation and paging mechanisms that are currently used. In addition, sections on microprogramming (3.8) and stack usage (3.7) discuss specific computer configurations.

2.1 Burroughs D825 System

Classification: Multiprocessor; ground-based military data-processing/real-time system

Operational Status: Operational

Description:

The hardware complement of the D825 may include from one to four processors, one to sixteen memory modules of 4096 48-bit words, and up to twenty I/O controllers. The modular organization of the system has been designed to achieve extremely reliable operation. Additional modules of each type may be added to provide redundancy; these elements do not remain idle, but share the processing load during normal operation. Burroughs claims that availability in excess of 99.99% can be achieved in this manner.

Communication between major elements flows through a distributed switching interlock of the cross-bar type. The switch

is designed so that no single failure can affect the operation of modules other than the one in which the failure is located. Memory modules may be used concurrently by all processor and I/O buses. Should two to more buses simultaneously attempt to access the same memory module, the switch resolves the conflict according to priority, and queues the lower priority requests.

The instruction set of the D825 contains instructions whose lengths vary from one to seven 12-bit syllables; zero-, one-, two-, and three-address formats are provided. Address modification may include infinite-level indirect addressing followed by indexing using one, two, or three index registers. A four-level thin-film operand stack is provided to reduce access time for repeatedly-used operands.

Software Characteristics:

The D825 Automatic Operating and Scheduling Program (AOSP) has three primary functions: it provides operational modularity to modular hardware, it provides system unity for real-time response, and it coordinates modules without the vulnerability associated with systems in which coordination is performed by a unit of hardware. A revealing statement is made in the second reference: "It is clear, however, that the D825 system would have fallen far short of the goals set for it if only the hardware had been considered. The AOSP is as much a part of the D825 system structure as is the actual hardware."

References:

1) "D825 Modular Data Processing System," Burroughs Corporation, Paoli, Pa., undated.

2) "D-825-A Multiple-Computer System for Command and Control", Proc. FJCC, 1962, vol. 22, Spartan Books, Washington, D.C.

2.2 IBM Direct-Coupled System (DCS) and IBM 360 Attached Support Processor System (ASP)

Classification: Dual computer system for ground-based general purpose data processing

Operational Status: Operational

Description:

The ASP and DCS systems are essentially similar; ASP utilizes two system/360 machines such as 65/40 or 65/50, while DCS typically consists of a 7044 and a 7094. The intention of the combination is to use the more powerful computer for the execution of the processor-limited part of each job, and to use the smaller for management of the I/O for each job and for job scheduling.

Communication between the computers is through I/O channels of each, which are connected by a channel-to-channel adapter, having the effect of causing each computer to look like an I/O device to the other. No sharing of core memory is utilized, although it is convenient to utilize secondary storage devices attached to both systems by means of two-channel switches.

Reference:

Rosin, R.F., "Supervisory and Monitor Systems", Computing Surveys, vol. 1, no. 1, March 1969.

2.3 Control Data 6600 and 7600

Classification: Multiprocessor; large scale ground-based general purpose data processing system

Operational Status: Operational

Description:

a) Central processor, plus peripheral processors (PPU).

b) Shared main memory.

c) Communication between processors via main memory and control instructions.

The CDC 6600 and 7600 are similarly organized, although the 7600 is a considerably higher-performance system. Each consists of a central processor and a group of smaller peripheral processors. The peripheral processors each possess private memory, but can address the system main memory as well. The intention embodied in the design of the system is that the central processor be devoted to the meat of the data processing job, while the peripheral units handle the I/O operations and clerical aspects. Thus, while the central processor is executing a program resident in main memory, one or more PPU's may be setting up another job in main memory for subsequent execution.

The central processor is itself designed to exploit some inherent parallelism in the sequence of instructions being executed: the functional execution elements are capable of independent and concurrent operations if operands are available and the logical constraints of the program allow.

To indicate the level of potential system performance, the following data is presented for the 7600: Cycle time for the 65K-word main memory is 275 ns; this memory is organized in 32 banks, which permits delivery of words at a rate of up to one per minor cycle of 27.5 ns. A 512K-word secondary core memory

is standard equipment, and is capable of delivering information at the same rate through use of an 8-word data path and 8-bank organization, even though its cycle time is 1760 ns.

Software Characteristics:

The 7600 will be available with an operating system to sustain the user in the remote batch, the time-sharing, and the real-time command and control environments.

Miscellaneous:

The attempt made in the CDC 6600 to exploit local opportunities for parallelism in the sequence of instructions being executed apparently was not as successful as the designers anticipated. This is indicated by two considerations: first, the CDC 6400 was designed to be like the 6600 except that instead of a concurrently-executing central processor composed of ten separate functional units, the 6400 had all functions combined into a single execution unit. The performance difference on typical benchmark problems was only a factor of two, however, showing that the achieved concurrency in the 6600 was markedly less than the possible concurrency. Second, the arrangement of function units in the 7600 has been significantly altered from that in the 6600, indicating that Contral Data believes that a better allocation of functions would improve the system performance. These considerations are mentioned here only to highlight the problems apparently inherent in exploiting, in hardware, parallelism which has not explicitly been identified in the program.

2.4 Univac 1108

Classification: Multiprocessor; large scale ground-based data processing system.

Operational Status: Operational

Description:

a) One to three processors (typical).

b) One or two I/O controllers (typical).

c) Up to four processors or I/O controllers may be attached to each I/O control unit.

Each processor can address all of main memory (which can be up to 262,144 36-bit words). Memory cycle time is 750 ns. Up to four logical banks for instruction/data fetch overlapping provide an effective cycle time of 375 ns., under control of the processor; up to eight-way interleaving is available to minimize conflicts.

Error detection in the hardware is limited to parity checking; each 36-bit word is provided with two parity bits. No arithmetic checking is performed. Storage protection is provided by means of a storage limits register which imposes strict boundaries on the areas of memory which may be accessed for instructions and data. A privileged mode exists where the limits are enforced only relative to writes, and an open mode provides free use of all memory.

Software Characteristics:

Exec-8 operational, but considerably behind schedule. Multiprocessor 1108 acceptance at Marshall Space Flight Center was delayed for two years due to software performance problems.

Miscellaneous:

Univac has prepared a formula for use in evaluating the performance improvement realizable from addition of processors to a system, and supplied numbers for the 1108 system. The formula is:

$$N = \frac{P \times 10^6}{C + Q + D + E}$$

where N is the instruction rate, P is the number of processors, C is the memory cycle time, Q is delay due to queues at memory units, D is the delay due to hardware (multiple module adapters, etc.), and E is the time added due to extended sequence instructions.

For one processor,

$$N = \frac{1 \times 10^6}{0.75} = 1.33 \times 10^6$$

With extended instructions,

$$N = \frac{1 \times 10^6}{0.75 + 0.30} = 0.95 \times 10^6$$

For two processors,

$$N = \frac{2 \times 10^6}{0.75 + 0.05 + 0.125 + 0.30} = 1.63 \times 10^6$$

Thus, the gain for the second processor is

$$\frac{1.63 - 0.95}{0.95} = 0.71$$

References:

1) Univac 1108 Multiprocessor System Description, Univac Data Processing Division, undated.

2) Stanga, D.C., "Univac 1108 Multiprocessor System", Proc. SJCC 1967, vol. 30, Thompson Books, Washington, D.C.

2.5 IBM System/360 Model 65 Multiprocessor

Classification: Multiprocessor; ground-based general-purpose data processing system.

Operational Status: Hardware and software operational

Description:

a) Dual processor system.

b) Shared main memory.

c) Shared I/O.

d) Direct communication between processors.

Each CPU can address all of the locations in main storage; each CPU has its own 4K byte interrupt area in main storage; each CPU can address any secondary storage device through alternate path I/O control; each CPU can reset, interrupt, or start the other CPU with a "Direct Control" signal or a "Malfunction Alert" signal through a direct hardware connection. The system can be reconfigured according to the availability of components. Optionally, the system can be run in the dual processor mode with main storage and I/O devices apportioned to the two processors.

Error detection in the hardware is the same as for the standard Model 65, and includes checking of arithmetic and logical operations as well as parity checking of information transfers. A group of "recovery management" programs attempt to recover from a machine malfunction by retrying the failing operation. If the operation cannot be retried, they assess program damage and either repair the effects of the failure or attempt to restrict the effects of the failure to a single job step. If the damage is unrecoverable, the job step is terminated. If the supervisor program is damaged, the system must be restarted. When "hard" errors occur, the operator is notified so that necessary reconfiguration may be accomplished. The operator may add or delete I/O devices, channels, CPUs, and blocks of main storage. However, at no time can any storage area containing a part of the supervisor be removed from the system.

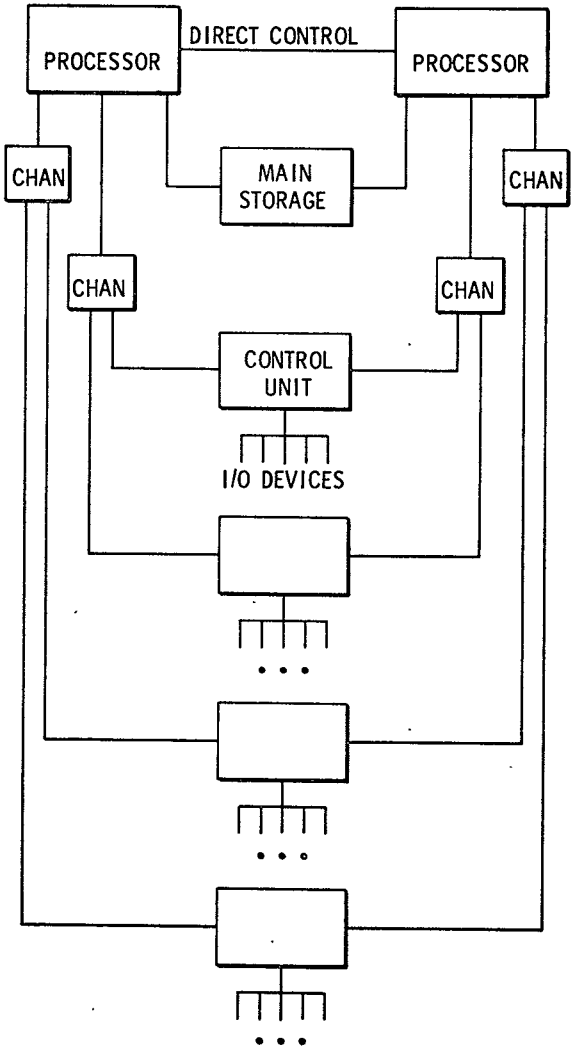


Figure 2.1 IBM 360/65 Multiprocessor

Software Characteristics:

A generalization of the standard OS/360 MVT is used. The fundamental unit of processing is referred to as a task; tasks are selected for execution by the supervisor routine, which searches a task control block (TCB) queue for the ready task with the highest priority. When a CPU discontinues execution of a task, the status of the task is recorded in the TCB. Task switching can take place as a result of a CPU interruption. If a CPU receives an interruption indicating that it should switch tasks, it does so. If the interruption indicates that the other CPU should switch tasks, it signifies this through a direct control signal which causes an external interruption in that CPU.

Miscellaneous:

Because of the design of the 360 I/O control units, connection to more than two processors is not feasible. Also, because of the nature of the direct control feature for communication between processors, the system is limited to two processors. Thus, the system is restricted by its hardware design from expansion beyond a two-CPU configuration, although the principles upon which the system is based are not so restricted.

References:

1) IBM Systems Reference Library, IBM System/360 Operating System, Model 65, Shared Main Storage Multiprocessing, Preliminary Description, Form C28-6671-0, Jan. 1968.

2) Witt, B.I., "M65: An Experiment in OS/360 Multiprocessing", presented at Information Systems Symposium, Sept. 4-6, 1968, Washington, D.C.

2.6 IBM 9020

Classification: Ground-based multiprocessor with graceful degradation capability, in a real-time application.

Operational Status: Operational

Description:

The IBM 9020 was built to meet the needs of the FAA's National Airspace System for air traffic control operations. One of the important requirements which the system was designed to meet was that of twenty-four hour fail-safe performance. To achieve this goal, a redundant group of substantially modified elements of System/360 Model 50 computers (later versions use 67 components) has been put together with a control program which is capable of directing error recovery in the event of a

subsystem failure. The resulting system embodies redundant capacity for all major elements, automatic error detection and dynamic system recovery capabilities, restart techniques for intermittent failures, and rescheduling of application functions when necessitated by solid failures. In the presence of a solid failure, the system can operate in a fail-safe mode by calling upon a redundant element; no functions are discontinued, nor are other aspects of system performance changed.

Should the number of available components fall below the number required to maintain complete performance, the system can continue in a fail-soft mode, with degraded performance, as long as at least one of each major element is operational.

The major components of the 9020 system are its memories, processors, and I/O control units. Six-port memories are used, connected to three processors and three I/O control units. Three tape controls are used, each of which is connected to two I/O controllers. Additionally, three peripheral adapter units are used as interfaces with external equipment. System units are classified operationally as active if they are involved in air traffic control operations, redundant when not so employed but available within a 30 second recovery period, or inactive if not in operational use nor available within 30 seconds. Redundant units may be used to expedite the repair of a malfunctioning unit, although they may have to be released to become active units in the event of another failure.

The configuration of the system is under program control; each of the five types of components mentioned above contain configuration registers whose contents may be set only by a privileged instruction, and only by certain processors, under control of the contents of the register itself.

Interlocking of common data in shared memory is accomplished by the standard 360 Test and Set instruction; however, a non-standard instruction has been added to allow a processor to delay a short time (for example, if it finds a desired memory area locked by another processor) without making further references to memory and possibly causing unnecessary conflicts.

Certain other non-standard instructions have been added to the 9020 system to enable a processor to identify itself and to set the location in memory of a preferential storage area itself. Others allow control of address translation which relates logical and physical storage locations in the system.

Software Characteristics:

The general control of program execution in the 9020 is quite similar to that of OS/360 MVT (multiprogramming with a variable member of tasks). Programs are divided into units, called tasks, which are scheduled for execution by priority. Single-level interruption is utilized; that is, when an inter-

ruption has control of a processor, further interruptions of that processor are inhibited. Continued processing of an interrupted task may be performed by another processor if one is available.

A problem encountered in both multiprogrammed and multiprocessing systems is that of avoiding mutual lockout because of the sequence and strategy for resource allocation. In the 9020, this problem potentially occurs with respect to allocation of main storage. It is avoided by defining a functional hierarchy for storage usage, and enforcing rules (in the control program) for storage assignment. The rules are that: 1) a task must request storage of different classifications in the order defined by the functional hierarchy; and 2) no storage of a given category may be requested by a task which has already been allocated, but not released or unlocked, storage of that category. Execution of a task which violates these rules is terminated by the control program. A task which requests storage not immediately available is suspended until the requirements can be satisfied.

Reference:

Entire issue, IBM Systems Journal, vol. 6, no. 2, 1967.

2.7 Univac AN/UYK-7 and 1832

Classification: Multiprocessor, real-time control computer. UYK-7 is primarily for Navy surface ships; 1832 is a new, faster, miniature version for the Navy ASW program, the S3A aircraft.

Operational Status: UYK-7 is operational; 1832 is in design phase.

Description:

a) 1 to 3 CPU's.

b) 1 or 2 I/O controllers, direct to memory independent of CPU.

c) Completely shared memory from 1 to 16 modules of 16 K words of 32 bits (max. size = 262,144 words).

d) 1 to 4 independent power supply modules.

e) CPU characteristics:

1) 16 and 32 bit instructions.

2) Data word size = 8, 16, and 32 bits (64 DP fixed-point).

3) Multiple accumulators with separate ones for executive mode.

4) Cascaded indirect and dual base and index addressing.

5) Instructions for 48 bit floating point data - 32 bit mantissa and 16 bit exponent (2^{32768}).

6) Memory lockout registers for memory protection.

7) Variable length character-handling instructions.

This computer is a large, flexible system. The instruction set is very extensive including immediate or literal and substitute types. The computer is reminiscent of an 1108 with a 48 bit floating point format.

No particular attention seems to have been given to failure detection and isolation or recovery procedures.

The multiprocessor software does not seem to exist but will have to be developed for the S3A program. The degree of difficulty should be approximately the same as for the 1108; in fact, the experience on that may be applicable. The progress of the S3A program should reveal the multiprocessing potentials of this design. The effort seems to address itself to the question, "Can a good, powerful, flexible, but conventional computer design lend itself to an efficient multiprocessor computer configuration without special M/P software-oriented hardware features?"

References:

1) Computer Data, AN/UYK-7, SB-12292, UNIVAC, December 5, 1968.

2) AN/UYK-7 (V) brochure, #PX 4758-A, UNIVAC Federal Systems Division, April 1968.

2.8 General Electric 645

Classification: Non-product-line time-sharing multiprocessor

Operational Status: Operational

Description:

The GE-645 is an extension of the 635, a product-line system. It was designed jointly by GE and MIT's Project MAC, a research

program sponsored by ARPA. The objective of the design was to produce a system which could run the Multics operating system, a comprehensive, general-purpose software research project intended to be capable of meeting the requirements of a large computer utility. The 645 differs from the 635 in the areas of the I/O controller, the interrupt structure, and most important, the addressing logic, which was the first to incorporate both segmentation and paging.

The system on the air at MIT is a two-processor configuration with 384K 36-bit words of memory. The novelty of the addressing is described further in Appendix A, but the part played by the processor will be briefly outlined here. Unlike more conventional processors, the 645 forms "two-dimensional" addresses by providing a base register which contains the number of an entry in a segment table, and computing separately the address of the word within that segment. The "procedure base register" holds the segment number for the procedure being executed; the "instruction counter" holds the offset. The "descriptor-segment base register" contains the address of the descriptor-segment or segment table; this address is added to the segment number to obtain the location of the appropriate entry. A register is provided to hold the segment number of the operand of the instruction, and four pairs of address base registers are used to hold addresses of argument lists, linkage segments, and stack segments. Eight index registers are included, and sixteen words of associative memory are used to contain recently-used segment and page table entries.

A special form of indirect addressing is implemented, which permits the generation of an interrupt (fault) when it is invoked; Multics utilizes this to provide a dynamic-linking facility in which linkages are completed as execution proceeds. This is useful in three ways, although it entails considerable overhead. First, it eliminates the necessity for a "link-edit" process prior to execution; second, it eliminates the linking of segments whose linkages are not used during a given instance of execution; third, it circumvents the problems caused by the fact that segment numbers are assigned during execution, not at compile time, and therefore cannot be placed into external addresses.

References:

1) Corbato, F.J., Vyssotsky, V.A., "Introduction and Overview of the Multics System", Proc. FJCC, 1965.

2) Glaser, E.L., Couleur, J.F., Oliver, G.A., "System Design of a Computer for Time-Sharing Applications", Proc. FJCC, 1965.

3) Organick, E.I., "A Guide to Multics for Subsystem Writers", MIT Project MAC Memos M0086, Nov. 1967; M0087, Feb. 1968; M0090, Feb. 1968; M0106, Jan. 1969; M0107, Feb. 1969; M0108, Mar. 1969; M0115, Aug. 1969.

4) GE-625-635 System Manual, Jan. 1965.

2.9 Honeywell Model 8200 (information mostly applicable also to H800 and H1800)

Classification: Two-processor ground-based data processing system, with hardware to permit concurrent multiprogramming.

Operational Status: H800, H1800 and H8200 operational.

Description:

Although the H8200 is a two-processor system (one word-oriented, and one character-oriented) with shared memory, the system is included in this summary because of the unique nature of its word-oriented processor, which is essentially the same as the H1800 and H800. Only the characteristics of this unit will be described.

Because of the unusual way the processor is organized, a brief discussion of two types of multiprogramming will be given. In a conventional multiprogramming system, a list of tasks is maintained, often ordered by priority. Normally, the processor is controlled so that the highest-priority "ready" task is being executed. When the task cannot proceed, for example because it is awaiting the completion of an I/O operation, it is removed from "ready" status and control is given to another task. When the operation-completion is signaled, the waiting task is placed in the "ready" condition again and execution resumes when its priority is the highest. Honeywell refers to this kind of processor-sharing as "vertical multiprogramming".

Another type of sharing, referred to by Honeywell as "horizontal multiprogramming" is implemented in the hardware of the H1800. A three-address instruction format is used, so that results are not normally left in the accumulator of the CPU between instructions. Although there is only a single copy of the arithmetic unit in the CPU, the program control registers (instruction counters, index registers, etc.) are replicated 8 times (9 in the H8200) so that up to 8 tasks can sequentially time-share the CPU on an instruction-by-instruction bases. A program control group which has been allocated to a task for execution is called an active group; the CPU scans for the next active group while it executes an instruction from an active group. When any task is awaiting completion of an I/O operation, a bit is set for that group in the hardware which causes it to be bypassed by the scanning process. Completion of the I/O operation causes this bit to be reset. When a group is not active or when it is stalled, as above, time slices which would have been used by that group are available for use by active groups.

The main advantages claimed by Honeywell for horizontal multiprogramming are that task switching is accomplished by hardware with zero time overhead, and combinations of programs that make heavy use of peripheral devices tend to obtain better overall throughput than under vertical multiprogramming.

Software Characteristics:

The characteristics described here are those of the Honeywell Mod 8 Operating System, written for the 8200. The 8200 has a ninth program control group, the Master Group, and a character-oriented processor (similar to the H4200 CPU), and thus this operating system is applicable only to the 8200, and not to the 800 or 1800.

Each job is assigned a priority by the user; the operating system selects jobs for execution based on their priority and their profitability. A job is considered "profitable" to run immediately if it requires the use of currently unused resources and does not also require the use of a currently busy resource that cannot be shared efficiently. Protection of programs from one another is accomplished largely in hardware; facilities employed for this purpose include memory protection, a peripheral protection/reassignment table, a privileged instruction set, and a watchdog timer to protect against endless interruptible loops. These features are augmented by software facilities which take cognizance of the detailed resource assignments.

References:

1) Hatch, T.F., Jr., Geyer, J.B., "Hardware/Software Interaction on the Honeywell Model 8200", Proc. FJCC 1968, Thompson Book Co., Washington, D.C.

2) Honeywell 1800 Programmers' Reference Manual, Honeywell Inc. Electronics Data Processing Division, Wellesley Hills, Mass., 1964.

2.10 Burroughs B6500

Classification:	Ground-base commercial general purpose data processing system with multiprocessor capability.
Operational Status:	Deliveries began in 1969. Software not complete.

Description:

a) 1 or 2 CPU's.

b) 1 or 2 I/O multiplexors.

c) Completely shared memory of up to 32 modules of 16K 48 bit data words plus 3 tag bits and 1 parity.

Burroughs commercial machines have for many years been unique. They have a radically different architecture and philosophy concerning the place of software. The 6500 merely continues these thoughts from their expression in the B5500. The B5500 has had an unusual reception among users. It has been a "later bloomer" in that demand has been increasing during each year of its existence, rather than being the largest when the machine was brand new. This has been true quite recently, even though the hardware was old and slow by comparison with newer designs. This growing group of enthusiastic users is quite a tribute to its unusual design concepts.

The B6500 is an attempt to provide up-to-date hardware to stay competitive and to increase the fold of satisfied customers. Rather than maintaining machine compatibility, they increased its capabilities in many areas such as memory capacity and I/O flexibility, while sticking to the basic philosophy that they have espoused before. The basic tenet seems to be, "Thou shalt not program in machine language". In fact, Burroughs does not even supply as assembly language.

They expect that all the programming will be done in a higher level programming language. The ones that they plan to give the chief support on the 6500 are:

1) ALGOL: Burroughs has used this extensively for many years.

2) FORTRAN, most widely used scientific language.

3) COBOL, most widely used business language.

4) PL/I, newer, but growing set of users.

Other features of the B6500 include:

1) Processor hardware design to implement higher level languages and run them under a comprehensive operating system called Master Control Program (MCP).

2) Multiprogramming is considered the normal mode of operation, and is recognized in the design.

3) 3 extra control bits (tag bits) in each word are used for flagging special characteristics.

4) A hardware stack mechanism is provided to automatically handle operand storage and other temporary data in a manner that makes it easy for compilers.

5) Polish notation type of instructions with variable number of syllables of 8 bits each.

6) Programs cannot be modified while in core. This produces re-entrant and even recursive subroutines as well as permitting automatic overlay with little special effort by the MCP.

7) Actual addressing is relative and/or indirect which makes it easy to relocate.

8) Memory protect includes upper and lower bound on arrays and descriptors for segmentation.

References:

1) Burroughs B6500/7500 Characteristics Manual, Burroughs Corp., Sept. 1968.

2) Burroughs B6500/7500 Electronic Data Processing System, July, 1968.

3) Hillegass, John B., "Burroughs Dares to Differ", Data Processing Magazine, July 1968.

4) Hauch, E.A., and Deut, B.A.,"Burrough B6500/7500 Stack Mechanism", Proc. SJCC, 1968, vol. 32.

2.11 IBM System/360 Model 195 (information below mostly applicable to Models 91 and 95 also)

Classification:	Single processor system embodying internal parallelism; very large ground-based general-purpose data processing system.
Operational Status:	Partially complete prototype operational; first delivery scheduled for 1971. Software operational. Models 91 and 95 operational.
Description:	Single processor system with extensive overlapping and "pipelining" of operations.

Although the M195 has only one processor, its unique degree of internal parallelism causes it to deserve consideration here. Five separate units may be operating concurrently; main memory, storage control unit and buffer storage, instruction processor, fixed-point/variable-field-length/decimal processor, and floating-point processor. Furthermore, each of these units may be performing several functions at one time. For example, as many as

three floating-point operations may be taking place concurrently.

A high speed buffer memory is used to partially mask the access time to main storage (810 ns). Additionally, an instruction look-ahead buffer is used to reduce conflicts between instruction and data word fetches, and to eliminate instruction fetching altogether for certain small program loops. Each of the two execution elements is provided with stacks to enhance pipeline operation. The floating-point add unit can deliver two 64-bit sums as often as every 162 ns; the multipy/divide unit can form a 64-bit product in 162 ns.

A pronounced degree of "real time" seeking of implicit parallelism is performed by the machine. That is, each instruction, after being decoded in sequence, is sent to an execution element where its further processing occurs sequentially only when expeditious or logically necessary. The use of buffer registers and other buffering techniques often makes out-of-sequence execution of instructions efficient. This philosophy is pursued to the extent that instruction decoding continues even in the interval between the point that a conditional branch has been decoded and the point that execution of the instruction which sets the condition code is completed. Of course, if the assumption made by the processor about whether the branch will or will not be taken proves false, the partially processed instructions must be canceled. However, as a hedge against this contingency, the instruction fetching mechanism fetches several instructions down the alternate path at the same time the conditional processing is taking place, so that regardless of the outcome of the condition test when it finally occurs, some progress has been made beyond that point in the program.

Software Characteristics:

A somewhat modified version of OS/360 is used in the M195. One of the consequences of the out-of-sequence instruction execution in the M195 is that certain interrupts are triggered after the location of the responsible instruction has been lost. The result is an uncertainty, for example, as to which instruction caused a storage-protection violation, overflow, etc., on some occasions. An instruction, otherwise a no-op, has been implemented in such a way in the M195 that no further instruction decoding takes place until the execution pipelines have been emptied. Although use of this instruction can prevent the uncertainty mentioned above, performance of the system is degraded since a great deal of the capacity of the processor is inherently disabled temporarily.

The design of the M195 clearly indicates an attempt to exploit parallelism implicit in ordinary coding prepared in the customary ways for a serial processor. However, the variation in performance between two versions of a problem coded with

and without M195 considerations can be quite dramatic.

Exceptional Characteristics:

Monolithic circuitry is used in the M195; basic stage delay time is less than 5 ns. Two boards of 8 x 12 inches hold pluggable cards which contain a floating-point add execution unit for 64 bits in which both preshifting and postshifting are accomplished. The high-speed buffer memory of 32K bytes is packaged on pluggable cards held by two 10 x 12 inch boards. The speed of the M195 clearly requires small physical size; its complexity, also required for high performance, tends to significantly add to the component count. Further increases in system performance will require comparable reductions in physical size, or more dependence on multiple processing, or both.

Reference:

IBM System Reference Library, IBM System/360 Model 195 Functional Characteristics, Form A22-6943-0, August 1969.

2.12 Control Data STAR Computer (String Array Processor)

Classification: Commercial data processing computer

Operational Status: Design Phase

Description:

This is a large 4th generation general purpose machine which is being designed for ground-based real-time applications including time-sharing. It is not a multiprocessor. It is being designed by a different team and is a complete departure from the architecture of the 6000 and 7600 series. General characteristics include:

1) Variable word length: 2-1024 bytes

2) Vector processors

3) 32 or 64 bit instructions

4) 32 banks of 16K 64-bit words

5) 32 and 64 bit floating point

6) 1000 data channels

The precise details of the computer are still somewhat tentative; it is anticipated that more information will become available when the design is frozen.

2.13 Hughes H4400

Classification:	Specifically a real-time multiprocessor development primarily aimed at military command and control
Operational Status:	Prototype hardware being built and software written by company funds. If funded, it could be a flight and/or a ground computer.

Description:

a) Up to 8 CPU's or I/O units total

b) Up to 16 banks of 16K 32 bit words

c) Central "cross-bar" switch that communicates and controls

d) Multiple usage registers for accumulators, index, and base registers

e) Various options allow capabilities to increase in the following order:

   1) 16 bit simplex, sequential machine

   2) 32 bit multiple memory, multiprocessor

   3) bit/string instructions

   4) floating point, SP/DP

   5) hardware macros, microprogrammed sequences (sine, arctan, etc.)

f) Hardware, interlocked multiprocessing executive

g) Memory protect but no memory paging

h) Special instruction for multiprocessing, e.g., interrupt assignment between processors

This is a computer development project to produce an expandable family of multiprocessors to meet various real-time computer needs. To do so, a multiprocessor hardware-software concept must be developed. They are proceeding along conventional lines with the addition of extra features to aid the multiprocessor executive problem.

Hughes has given a good deal of thought to the failure detection and isolation problem. They estimate that 90% of the failures can be diagnosed down to the "card level". They

also have devised a system of switching in and out various CPU's and memory units after the failure detection. This is done by hardware in an extensive switching unit. A system has been devised to provide for failure of all modules including an executive processor or memory. The only function not neatly handled is the manner in which the programs are resumed via a "roll-back". They consider this to be "an applications programming problem". This leaves it up to the software to do it which may prove quite difficult.

Although the computer is still in the prototype stage of construction, extensive software is being developed. This includes:

1) A meta-assembler to attempt to allow for compatibility between configurations.

2) A simulator to run on the Control Data 6600.

3) A JOVIAL compiler.

4) A run time package that includes a real-time multiprocessor operating system and library and utility routines.

2.14 Safeguard Central Logic and Control Computer (CLC)

Classification: Ground-based multiprocessor

Operational Status: Two development models in operation

Description:

The design of this machine began in 1964 for the Nike-X system; it has also been through the Sentinel phase on the way to becoming the Safeguard Computer. Designed by Univac and built by Western Electric, the system can have up to 10 processors and 16 "program" and 16 "variable" memories of 16K 64-bit (plus 4 parity bits) words. Program memories, which originally were planned to be ROM's, now are similar to the variable memories except that they may be written into by I/O but not processors, and they have two access units per module rather than one. Sixteen I/O channels are used to communicate with standard peripherals plus the two radars and mission-oriented command and control equipment.

The system functions as a special-purpose controller much like a missile computer; little use is made of interrupts, and a fixed pattern of computing is performed, with the major cycle determined by the characteristics of the phased-array radars. (Radar beam-steering is performed in special computers located at the radars, and not by the CLC.) One spare copy of each type is maintained on-line to serve as a replacement in

the event of failure. Parity checking, but no arithmetic checking, is performed. Each target is computed by a seven-element Kalman filter; the seventh is the ballistic coefficient. If track is lost or memory errors occur, the filter calculations can be started over, making recovery from failures rather easy. Missile guidance computations apparently can similarly be restarted.

To reduce the frequency of memory conflicts, some units of program are replicated in other modules. The speed of the memory is 0.5 μs; however, cable length is substantial, causing propagation delays to be appreciable. Examples of register-register operations speeds are: add, 0.2 μs; multiply, 0.53 μs; divide, 5.9 μs. Approximately 1.5 million instructions per second are performed.

An advanced design being considered includes the addition of two array processors to the system. One would be used for the tracking function, and one for guidance, with a processing element assigned to each individual target or missile.

2.15 IBM 4-Pi Model CP-2

Classification: Airborne real-time flight computer

Operational Status: CP-2 is fully operational with flying hardware and software.

Description:

a) One CPU

b) Two I/O channels tied to CPU

c) 8K to 32K 32-bit words

d) CPU characteristics:

1) 16- and 32-bit instructions

2) 16- and 32-bit data words

3) Single accumulator with extension register

4) Three index registers. One is hard wired, two in memory

5) Eight interrupts on two different levels

This is an older computer and not as advanced in features as some of the others, but it has benefited from the wealth of experience gathered through its widespread use. The hardware

is readily available and there is a great deal of good support software.

The computer does not lend itself to a multiprocessor configuration, because it is not possible to share memory. However, it has been used in a federated dual computer mode in the F-111 Mark II Avionics Computer System. This uses two parallel computers with separate memories, but with the ability to send data back and forth to one another. One computer is the general navigation computer and the other is a weapons delivery computer. Key variables when computed by either are transmitted to the other one. Programming was done in such a way that if either computer failed, the other would be able to carry on alone and execute the important jobs with degraded performance.

References:

1) System 4-Pi CP-2 Technical Description, IBM Electronic Systems Center, Owego, N.Y., Revised May 1969.

2) System 4-Pi Model CP-2 Support Software, IBM Electronic Systems Center, Owego, N.Y., Revised August 1968.

3) Daggett, E.H. and Lee, R.Q.,"The F-111D Computer Complex", General Dynamics Corp., Fort Worth, Texas, AIAA Paper No. 68-837, August 1968.

2.16 IBM 4-Pi EP/MP Computer

Classification:	Airborne real-time computer, multiprocessor configuration
Operational Status:	Hardware is operational. Computers were delivered for MOL but project was cancelled. Multiprocessor configuration developed for VS A-NEW, the Navy ASW research project at Johnsville, Pa. Software is now being being prepared.

Description:

a) One or two CPU's (three CPU's are possible)

b) Two HIMAC (high speed multiplexer and control units), the main I/O controllers

c) Up to eight modules of 16K 32-bit words with four ports, one for each of the CPU's and HIMACs

d) CPU characteristics:

1) 360-compatible instruction set (16, 32, and 48 bit instructions) with the addition of special microprogrammed instructions - sine, cosine, arc tangent, and square root. Floating-point instructions are an optional extra.

2) 32, 16, 8 bit data words

3) 16 registers employable as accumulators or index registers

4) Extensive instruction set including execute, move, and binary-to-decimal conversion

This is the only computer in the 4-Pi set that is compatible with the ground-based 360 series. By using the same 360 programming architecture for a flight computer, it is presumably possible to do a detailed check-out and simulation on ground equipment that will verify the programs to be used in the flight computer.

The emulator system allows one to run EP programs under Operating System/360 in either a direct or interpretive mode. The interpreter mode simulates the instruction and permits detailed evaluation via full traces, snapshots, and detailed timing information on an instruction-by-instruction basis. This presents a full history of the operation for debugging purposes. The direct mode of the emulator system simulates the execution of the EP by executing the EP program directly on the system 360 in so far as possible. Only the instructions unique to an EP are executed interpretively. This produces a very fast simulation, but loses the capability of tracing and detailed timing information.

The VS A-NEW project is developing a dual-processor multiprocessing system consisting of two EPs. Dual-processor operation proven on the System/360 Model 65 shared storage multiprocessor system has been incorporated into the VS A-NEW software. It includes a floating executive that can be run on either processor. Not only can the two processors work independently on separate problems, but it is hoped that they can work cooperatively on the single problem that requires extra high-speed processing.

Reference:

VS A-NEW Brochure, IBM Federal Systems Division, Owego, N.Y., Brochure #69-825-1A.

2.17 Litton L-304, 305, 3050, 3070

Classification:	Multiprocessor, real-time control flight computer family
Operational Status:	Dual processors used in E-2B and E-2C, Navy Airborne Warning System. Hardware is operational. Single 3070 proposed for AWACS

Description:

a) 1 or 2 CPU's

b) Up to 8 I/O stations and up to 64 channels to transfer data simultaneously

c) Shared memory of up to 16 blocks of 8192 words of 32 bits (max. size of 131,072 words)

d) CPU characteristics:

1) 32 bit instructions

2) 16 or 32 bit data

3) 64 program levels with automatic priority queueing and

4) 8 multipurpose registers for each level.

5) Variety of addressing modes

6) Many real-time clocks with interrupt for each

7) Comprehensive instruction set including MOVE, EXECUTE, EXCHANGE, and good literal handling, but no floating point

This computer has a large capability and seems to be a well thought-out design. Its unique aspects are centered around the 64 different program levels and built-in hardware type of executive, and the tie-in of interrupt structure to this executive.

Some thought has been given to failure detection. In particular, a system has been designed that allows one processor to run as a back-up to another with automatic switch over in case of failure.

The E-2C has two processors working on different dedicated job streams with the capability of either doing the important tasks in case of failure. The software has not been fully worked out.

For AWACS, the initial proposal was for three L-3050's - two working steadily and one on standby or assigned to low-priority tasks. But the multiprocessing problems appeared complex enough to tip the scales in favor of a newer, faster version (L-3070) to do the job in a simplex mode of operation. The conclusion may be drawn that, in spite of the fact that the machine was designed for multiprocessing, the problems of producing a large-scale cooperative multiprocessing system (in particular, the software) are severe enough that a reasonable alternative is preferred. This is undoubtedly an oversimplification and may exaggerate the situation, but Litton's belief seems clear.

Additional characteristics of the larger machines in the family (L-3050, 3070) are:

1) Memory paging and memory protect features

2) More powerful instructions including floating point options, substitute, test and insert/skip

3) 16 registers per program level

4) Special linkage and level registers

Reference:

Litton L-304 System Application, Litton Data Systems Division, Van Nuys, California, July 17, 1967.

2.18 ERC EXAM Computer

Classification: Flight multiprocessor

Operational Status: Early design, in hardware development stage

Description:

The chief area of this effort at ERC has been in the design of the modular cross-bar switching network, which connects the individual memory modules to the various processors. The logic is such that while one processor is connected to particular memory module, other processors may be simultaneously communicating with other memory units. When a processor needs to access memory, its request is sent to the appropriate memory module. If the memory is not busy servicing another processor, it will grant a request for a new memory access. Simultaneous requests from different processors to the same memory unit would be resolved on the basis of priority. The big advantage of the cross-bar scheme is the possibility of simultaneous data communication between two or more processors and memory at the same time. A

much higher theoretical data transfer rate is possible. This cross-bar is modular to allow expansion of more memory or more processor units. Also proposed is a floating executive control.

References:

1) Wang, Gary Y., "An In-house Experimental Air Space Multiprocessor - EXAM", ERC Memo #KC-T-031, September 20, 1967.

2) Wood, Paul E., Jr., "Interconnection of Processors and Memory in the Multiprocessor System", ERC Memo #KC-T-041, February 5, 1968.

3) Wood, Paul E., Jr., "Input/Output System for An Aerospace Multiprocessor", ERC Memo #RC-T-062, May 19, 1969.

2.19 MIT/IL ACGN Computer

Classification: Aerospace multiprocessor designed for graceful degradation

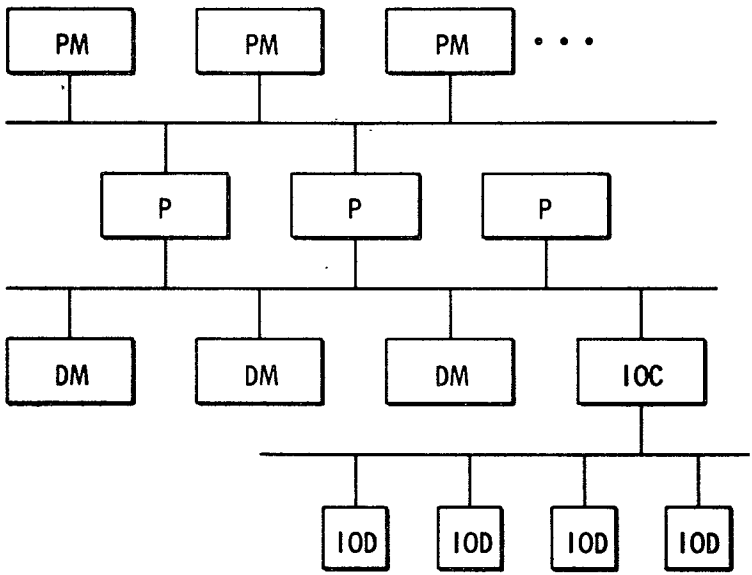
Operational Status: Paper design only

Description:

This computer is a multiprocessor design based on the anticipated requirements of a control, guidance, and navigation job which is "advanced" relative to the Apollo mission. Several types of experience on Apollo have contributed to the system: I/O rates involved; reliability; processing speed; programming ease; expandability.

The design of the system was not completed because of exhaustion of contract funds. As a result, a number of loose ends exist; however, most of the architectural considerations were specified. A three-bus system was chosen: one bus connects the processors and program memory, one connects the processors, I/O controller and data memory, and one connects I/O devices and I/O controller. A serial data bus was chosen because of its conceptual simplicity and consequent reliability, and because of the apparent difficulty in designing a gracefully expandable crossbar array. The serial bus technique offered the expansion potential of simply attaching additional modules of the desired type to the bus; so long as the bus capacity is sufficient, the system may continue to grow.

The concept of graceful degradation was realized by planning the use of more processors than required to accomplish the computational functions, in conjunction with a software system which could recover from the loss of a processor at any time. Memory failures were to be masked by creating extra copies, in separate modules, of critical data, so that no failure of any



PM: PROGRAM MEMORY MODULE

P: PROCESSOR

DM: DATA MEMORY MODULE

IOC: I/O CONTROLLER

IOD: I/O DEVICE

Figure 2.2 MIT/IL ACGN Computer

module was capable of preventing recovery of such information. Obviously, in such a system, combined consideration of hardware and software aspects was required to achieve a viable design.

The desire for continued operation in the presence of failures necessitated certain design characteristics with respect to error handling.

1) All components of the system are to be infallible under error detection; that is, the probability of the occurrence of an undetected failure must be negligible.

2) Certain components of the system are to be infallible under error correction; so that the probability of a non-masked error in such components is at least as small as the probability of an undetected error in a fallible component. Components required to be infallible in this way include the buses and their associated logic, program memory, and the I/O control unit.

3) Pages in data memory may fail in a detectable manner; however, since critical data may be replicated in more than one memory module, data may be considered to be infallible even though individual memory modules are not.

4) Fallible components which have failed must be capable of being isolated from the system.

Software Characteristics:

Executive control of tasks in the system is of course a key function. Because of the high traffic which might be anticipated in the executive process, two parallel approaches were followed by MIT; selection of one over the other did not take place, and might, in fact, depend upon the particular application of the system. In one approach, a special purpose system module was provided to perform the executive function. This module, attached to the system data bus, would contain processing elements and memory intended to remove most of the executive data flow from the data bus. In the alternate approach, the executive function was performed entirely in software, avoiding the need for a special module with its replications to assure infallibility. In both approaches, the functions performed would be similar; the design proposed for the software executive will be briefly described here.

Executive control is centralized around several lists of data and a multipurpose special register located in the I/O control unit. Only two of the lists will be mentioned here: the dispatch list and the wait list. The dispatch list contains all active requests for processing, ordered by priority and age. The wait list contains pending requests for processing which are to be issued at specified times. The special register contains both the location in the dispatch list of the next

process request to be honored, and a group of bits whose purpose is to indicate the occurrence of certain external events which in some systems might trigger interruptions. It was believed in the ACGN computer design, however, that computations would be divided into several-millisecond sections, referred to as jobs, and that this level of division would be the one at which competition for processors took place. Thus, in many a processor system, the average interval between a given instant and the time at which some processor next completed a job would be small. Thus, interrupts and their associated overhead could be avoided by having each processor check for the presence of an unserviced external event prior to taking the next active job from the dispatch list. If one or more such bits were present, the required functions would be performed by the discovering processor.

Provision of a compiler for the system was planned. Not only was this approach felt to be important from the ease and speed of programming point of view, but the use of the compiler as a program-convention enforcer seemed equally desirable. Because of the constant problem of multiprocess interference when common data is involved, some kind of interlocking is necessary. To reduce the number of ways or occasions when it would be possible for a programmer to inadvertently misuse the protection mechanism, the compiler could be equipped to do virtually all of the interlock administration.

Reference:

"Control, Guidance, and Navigation for Advanced Manned Missions", MIT Instrumentation Laboratory Report R-600. Vol. 2, Cambridge, Mass., January 1968.

2.20 ERC-Hamilton Standard Modular Computer

Classification:	Replaceable modular flight computer (MFC)
Operational Status:	Prototype hardware version completed and delivered to ERC

Description:

A computer consists of one of four types of units. They are a MU (memory unit), CU (control unit), AU (arithmetic unit), and an IU (I/O unit). Supplied are several of each type of unit and a master switching unit that selects the modules that are active. This is called the CAU (configuration assignment unit) and is responsible for maintaining a set of operational modules.

Diagnostic programs are used to detect and isolate mal-

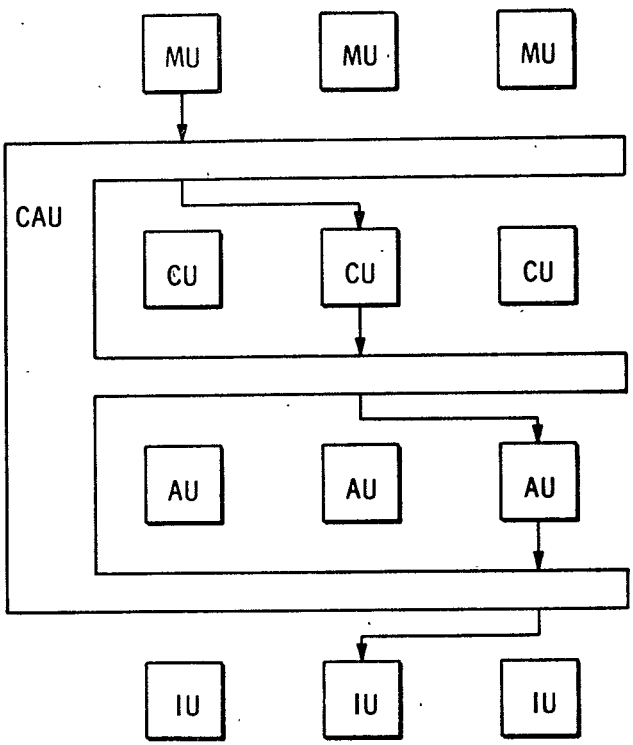


Figure 2.3 ERC-Hamilton Standard Modular Computer

functions in the units. Fault detection circuits can initiate the diagnostics. They will categorize the failure and ask for new units from the CAU if the failure is not a transient one.

The distinguishing feature is that this is not a multiprocessor configuration, but two or three separate computers. Possible modes of operation include the following:

1) Three computers working on the same problem and voting on their results during a high reliability period such as boost.

2) Three computers working on independent problems.

3) A method to keep several computers on-line with a minimum of spares.

The system has possibilities but is complex because of the amount of switching hardware needed. It also necessitates an infallible CAU to achieve the reliability goals. It seems of dubious value compared to an equivalent multiprocessor. For another approach towards the same goals, see the JPL STAR computer.

2.21 MIT/IL SIRU Computer

Classification:	Simplex computer with spare units for automatic backup
Operational Status:	Breadboard under development

Description:

This computer has been designed by the MIT Instrumentation Laboratory as part of the Strapped-down Inertial Reference Unit (SIRU) system. The major function performed by this machine is the maintenance of the quantities which describe the inertial attitude of the inertial subsystem via measurements incorporated every ten milliseconds. Additionally, the computer calculates velocity from accelerometer measurements, and has several milliseconds left over to devote to other jobs.

The computer contains two processors and two memories. One processor is kept in a standby condition while the other operates. Error detection features throughout the processor are provided to signal the occurrence of single errors. If an error is detected, the active processor will initiate turn-on of the standby processor and concurrently attempt a re-try of the current instruction. If the re-try is successful, however, the turn-on of the standby unit is terminated.

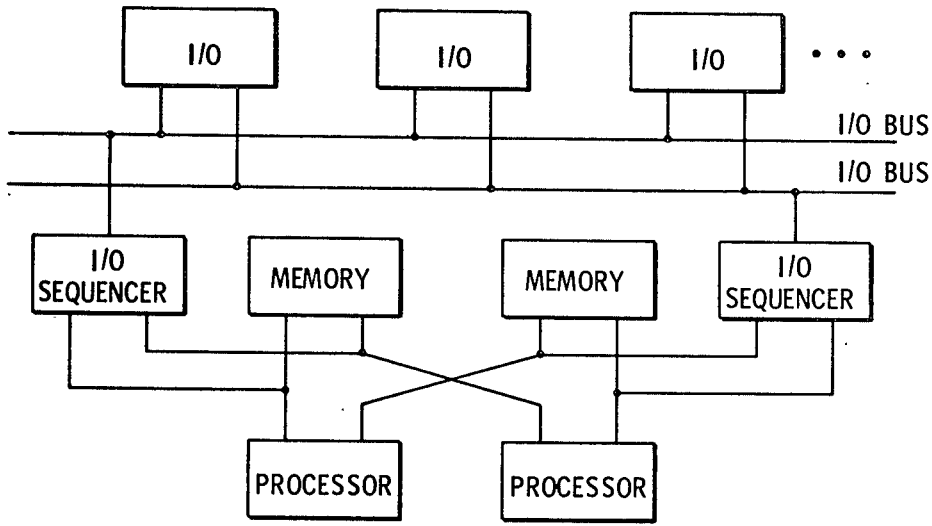


Figure 2.4 MIT/IL SIRU Computer

Operation of the memories is somewhat different: all data written into memory is written into both, so that normally both units contain identical data. Data read from memory, however, comes only from the unit currently designated as the active one of the pair. Should an error be detected in the active unit, the memories exchange roles and operation continues. Operation of a duplexed I/O system follows a similar pattern.

Exceptional Characteristics:

Each memory unit in the SIRU computer contains a high-speed scratchpad and working-register storage section. Execution of instructions in the processor has been separated into two distinct parts: first, the computation is performed and the results stored in dedicated area of high-speed memory; second, these results are moved from their temporary locations to their final destinations. The advantage of this technique is that each part of the execution of an instruction may be safely re-initiated after partial completion since neither of the two parts stores any results in locations occupied by the operands for that part. Since the working registers for the processors appear in both memories, any instruction execution which suffers a fault in either working register or processor may be completed or redone using either the alternate memory or the alternate processor, or both.

This inherent ability of the hardware to perform successful error recovery in a manner totally transparent to the software causes some sacrifice in processor speed. However, it eliminates both the necessity for failure recovery software and the historically knotty and costly effort required to verify the adequacy and accuracy of such coding. This is felt to be an extremely significant step in coordinated hardware/software design.

Reference:

Crisp, R., Gilmore, J.P., and Hopkins, A.L., Jr., "SIRU - A New Inertial System Concept for Inflight Reliability and Maintainability", MIT Instrumentation Laboratory Report E-2407, May 1969.

2.22 JPL STAR Computer

Classification:	Experimental aerospace computer with built-in automatic maintenance features. Not a multiprocessor.
Operational Status:	Experimental prototype under development; several subsystems have been completed and tested.

Description:

The JPL STAR (self-testing and repairing) computer has been designed as an attempt to provide an error-free, unattended computer system which could operate for several years during the unmanned exploration of the solar system. The principal system features used to diagnose and recover from errors are:

a) Use of error-detecting codes to allow fault identification concurrently with program execution.

b) Subdivision of the computer into a number of replaceable functional units.

c) Fault recovery carried out under the control of special-purpose hardware; consists of program repetition or replacement of faulty units.

d) Unit replacement accomplished by power switching; information lines of all units are permanently connected to the busses through isolating circuits; unpowered units produce only "zero" outputs.

The functions often implemented in the CPU of a computer have been split into five subunits: the main arithmetic processor, the logic processor, the control processor, the timing processor, and the interrupt processor. Except for the logic processor, which runs with two copies operational, for checking, only one copy of each unit is powered, and several unpowered backup copies are provided. Upon sensing of an error, the test-and-repair-processor (TARP), a processor unique to the STAR computer, directs the recovery operations. Because of the key role played the TARP in error recovery, three powered copies of the TARP are run concurrently, with outputs determined by voting logic.

If a powered TARP disagrees with a voted output, it is immediately returned to the standby condition and power is applied to one of the other standby units.

Software Characteristics:

The software design for the STAR computer is only partially complete. A key aspect of this software is the ability to perform a "rollback" to a previous point in the program as part of the error recovery process. Although an instruction has been provided which stores a "rollback" address in the TARP for this purpose, it appears that any attempt to incorporate multiprogramming into the system will necessitate use of a group of such addresses (viz., one for each active or scheduled task) plus other information for rollback purposes. Although multiple copies of this data would be necessary as a protection against memory loss, the current configuration of the system requires

storage of this data outside the TARP.

Miscellaneous:

Because error detection is such a crucial component of the system, a brief description is included here. Operand words consist of eight 4-bit bytes, one of which is a check-byte whose value is 15 minus the modulo-15 residue of the value of the other seven bytes. The checking algorithm computes the modulo-15 residue of the entire operand word; a nonzero residue indicates a fault.

An instruction word consists of a three-byte operation code and a four-byte address. The eighth byte is used as a modulo-15 check on the four address bytes; checking of the op-code consists of verifying that exactly two bits of each byte are ones. In some cases it is necessary to perform both checks for validity on a given word and rule it fault-free if it passes either one.

Residue bytes are processed independently in the arithmetic processor to provide a check on the arithmetic processing itself. Because the residue byte propagation in logical operations is difficult to compute, however, two copies of the logic processor are operated concurrently, and the outputs are compared to verify accuracy.

References:

1) Avizienis, Algirdas, "Design of Fault-Tolerant Computers", Proc. FJCC, 1967, vol. 31, Thompson Books, Washington, D.C.

2) Avizienis, A., Mathur, F.P., Rennels, D.A., "Automatic Maintenance of Aerospace Computers and Spacecraft Information and Control Systems", AIAA Paper No. 69-966, AIAA Aerospace Computer Systems Conference, Los Angeles, California, Sept. 1969.

2.23 RCA 215

Classification: Airborne real-time multiprocessor

Operational Status: Under development

Description:

a) 1 or 2 CPU's

b) 1 or 2 I/O units

c) Main memory of 2 to 8 modules of 16K 32-bit (plus 4 parity) words (64K bytes/module)

d) Expanded version offers up to 4 CPU; 4 I/O units, and 16 memory modules (1 megabyte maximum)

e) CPU characteristics

1) Instruction set is fully compatible with the Spectra 70 and the non-privileged ones of the IBM 360.

2) Scratchpad storage consists of 64 words of 36 bits each with 300 ns cycle time.

3) 1024 64-bit words of ROM used for microprogram storage. Cycle time is 300 ns.

4) Automatic fault diagnosis and error recovery.

5) 4 processor states with special registers for each. 32 priority levels of interrupt use three of these states.

This recently announced airborne computer system is another one that offers compatibility as one of its chief virtues. As its manufacturer states, in order to supply the extensive and complex functional programs and support software that is needed, it is desirable to capture the work done on existing commercial software systems. To attempt to develop a complete software package for a special military application is extremely costly, in both time and money. The solution according to RCA is to rigorously produce a flight counterpart to a commercial computer. It is not sufficient to imitate a ground computer by implementation of only a subset of the instructions or generating results which are "nearly the same". The flight computer must duplicate the ground-based version on a bit-by-bit basis including non-instructional features. As a bonus, the ground twin can be used for support for compilation and checkout.

With this aim, RCA has produced a computer that contains the entire instruction set of Spectra 70 series of computers (35,45,55) including the privileged instructions. It also duplicates the four program states, the I/O channel control, the interrupt management scheme,and other features of the Spectra 70. As a result, any user program compiled and tested on a Spectra 70 will run without alteration on the 215. The 215 has added instructions used by the executive for control of multiprocessing and error recovery. This same instruction repertoire (as in the Spectra 70) is fully compatible with the non-privileged mode of the IBM System 360. This opens the door to a vast collection of existing programs that would operate on the 215.

Another area that has been emphasized is that of fault tolerance and error recovery. RCA has conscientiously striven for a fail-soft computer complex. They have made a rigorous attempt to avoid "single-thread" hardware and attain the capability for "graceful degradation" while running programs written

for a different family of computers. Towards this end, hardware checking and other features have been incorporated as well as extensive software routines in the executive including items such as a "recovery nucleus" in a separate memory module. The degree of success of these measures is not easy to ascertain, but the underlying motives should be highly praised.

References:

1) Dieterich, E.J. and Kaye, C.C., "A Compatible Airborne Multiprocessor", FJCC, 1969, vol. 35, pp. 347-357.

2) "Introducing the RCA 215 Military Computer", RCA Aerospace Systems, DEP/SCN 101-69.

2.24 Control Data ALPHA

Classification: Airborne computer

Operational Status: Operating prototype has been demonstrated

Description:

This is a proposed LSI implementation computer. Features include the following:

a) Up to 4 CPU's or I/O units total

b) Up to 8 banks of 16K 32 bit words

c) CPU characteristics:

   1) 16 and 32 bit instructions

   2) 16 registers for accumulators and index registers

   3) Floating point instructions, SP and DP

   4) 32, 16, 8 bit operand instructions

   5) Special trig function instructions - sin/cos, vector rotation, square root, rectangular to polar conversion

   6) String and search instructions

Reference:

Control Data Brochure, "ALPHA Computer Family", #100, 644B.

2.25 Litton IRAD

Classification: Multiprocessor

Operational Status: 1975 target date

Description:

This system is at present a paper design of a computer organization suitable for use as either a flight or ground computer. It is a company-funded effort, with three major goals:

1) Efficient multiprocessor structure

2) All LSI, for reliability and size

3) The instruction set is to efficiently use memory

Litton has strongly attacked the third of these, in the belief that the extra cost of logic required to implement powerful instructions will be significantly less than the cost saving achieved through improvement in memory utilization. The instruction set design at this time is claimed to require only 40% of the number of instructions used to code a similar problem mix for the Litton 3050, and only 49% of the bits.

The instruction set differs from conventional sets in that it is strongly oriented to the processing of bit fields, rather than bytes or words. The arithmetic or general purpose registers of the machine have been designed to reflect these considerations. Data in registers is held in a floating-point format, with a 40-bit mantissa and an 8-bit "power". The mantissa is not usually normalized; a type of "significant digit" arithmetic is performed which preserves available accuracy but requires less time to execute. When bit-fields are fetched from memory to registers, the "power" field is specified in the instruction, rather than by the data itself. Similarly, the scaling for a store order is also contained in the instruction. Because of use of push-down mechanism for register addressing and elimination of the indexing field when not needed, Litton claims that the average instruction length is about normal, even though field and power data is included when needed.

The implementation of multiprocessing is accomplished using an adaptation of the 64-level program hierarchy introduced in the L304. Major changes include modifications to the reserved-memory area to reduce the number of unused locations, addition of storage interlocking machinery, and extension of the program-level switching logic to facilitate multiprocessing. A four-tier storage hierarchy is used: program, local, compool, and "multi-level" data areas are recognized.

This machine embodies some of the most novel ideas we have encountered.

2.26 Burroughs Interpreter Computer

Classification: Airborne multiprocessor

Operational Status: In preliminary design phase

Description:

Unique features include:

1) Two levels of microprogramming - referred to as micro and nano programming (one level wired in, the other loadable from memory).

2) This allows the flight computer to look like any computer that might be desirable, e.g. an IBM 360 or a B5500.

Reference:

Advanced Multiprocessor Computer Development, Burroughs Corporation, OS SSG, August 5, 1968.

2.27 U.S. Navy NAVAIR AADC (Advanced Avionics Digital Computer)

Classification: Generalized family of real-time flight computers

Operational Status: Paper computer - preliminary design phase

Description:

The AADC program is attempting to develop a general purpose modular set of digital computers to meet the Naval Airborne Computer requirements for the 1975-85 time frame, using the building block approach. The fundamental goal is the feasibility of the design of a spectrum of computers from the same basic functional and byte-functional elements. This will hopefully allow the reduction of the development cycle time from years to a matter of weeks. The building blocks take advantage of LSI and MSI technologies. The availability of these building block modules will permit the rapid configuration of an airborne digital computer system to meet a given set of specific operational requirements. A big problem is determining a modular organization of the computer. The organization must be general and powerful enough to satisfy the most exhausting performance requirements that can be projected. At the same time it must be divisible into smaller units needed to handle less demanding tasks in a cost-effective manner.

It is expected that the computer will be microprogrammed to provide an AADC with a variable instruction repertoire as well as the capability to emulate computers already in the Navy inventory. The heart of the AADC approach is the byte-functional module. This is proposed for its flexibility. It allows variation in the specific computer organization and permits the computer word length to be chosen for a particular application to meet the specific requirements. Because each specific computer developed from these building blocks might be substantially different in its structure, the AADC program has proposed a meta-compiler to accompany this set of computers. This compiler could be adjusted to suit each unique hardware organization and instruction set. The alternatives of identical computers for all applications or the creation of a new compiler for each hardware design are considered too restrictive.

The Navy also envisions the establishment of a data bank for "best case" computer algorithms for solution of many of the common computational problems.

Reference:

"Advanced Avionics Digital Computer Base-Line Definition", Report #AIR-53333Fa, Naval Air Systems Command, Washington, D.C., 23 July 1969.

2.28 SOLOMON

Classification: Parallel network computer (experimental)

Operational Status: Unknown

Description:

SOLOMON consists of a 32 x 32 array of processing elements (PE's) under control of a central control processor. The central control unit contains program storage, has the means to retrieve and interpret the stored instruction, and has the capability, subject to multimodal logic, to cause execution of those instructions within the array. Thus, at any given instant, each processing element in the system is capable of performing the same operation on the operands stored in the same memory location of each PE. Because each PE is provided with its own core storage unit, these operands may all be different.

Each processing element may communicate with its four adjacent "neighbors". The "edge" elements, which do not possess a full set of neighbors, use their free connections for I/O. Additionally, the central control may broadcast constants for use by all members of the array.

Each PE in the array has a mode register; commands from the central control to the PE are executed by the PE only when the mode signals from the controller match the mode stored in the PE.

Reference:

Slotnick, D.L., Borck, W.C., and McReynolds, R.C., "The SOLOMON Computer", Proc. FJCC, 1962, Spartan Books, Washington, D.C.

2.29 ILLIAC IV

Classification:	Parallel-array computer; contains 256 processing elements (experimental)
Operational Status:	Under development; target is late 1970.

Description:

The ILLIAC IV structure consists of 256 processing elements (PE's) arranged in four arrays of 64 processors each. A thin-film memory of 2048 words is provided with each processor. A common control unit for each array decodes the instructions and generates control signals for all processing elements in that array. A central index register group is included in the control processor, and an index register and address adder is provided in each processor for independence of operand addressing. Each processor has an enable flip-flop whose setting controls that unit's instruction execution. This bit is part of a test-result register in each PE which holds the results of tests on local data.

Data routing the processors is provided by connections to units i+1, i-1, i+8, and i-8 from each unit i; end-around connections are provided for "edge" processors. (See Fig. 1.8)

The four arrays may be operated independently, in pairs, or all together. The end-around data routing connections are modified when the array configuration is changed. The system program resides in a Burroughs 6500 general-purpose computer, which supervises program loading, array configuration changes, and I/O operations internal to the ILLIAC IV system and to the external world. A large disk storage system is directly coupled to the arrays, and there is also a provision for real-time data connections to the arrays.

Instructions belong to one of two classes: control unit (CU) instructions and PE instructions. The former control the addressing and sequencing in the CU, while the latter are de-

coded in the CU and then transmitted to all the PE's.

Software Characteristics:

The ILLIAC IV operating system resides in the B6500, and uses the standard B6500 master control program (MCP) for processing of most tasks.

The system designers have decided that for effective use of the parallel array elements, it is essential that all possible parallelism be detected in those algorithms which are to be executed. They have further concluded that the difficulty of achieving this if the algorithms are specified in languages such as FORTRAN or ALGOL is prohibitive. Thus they have designed a language, TRANQUIL, which is intended to allow the user to express array-type computational processes in terms of arrays and parallel operations. A key feature of the language is its mapping function, used to map arrays to optimize data transfers between primary and secondary memory, to minimize unfilled areas of primary memory, and to optimize the use of the PE's.

References:

1) Barnes, G.H., Brown, R.M., Kato, M., Kuck, D.J., Slotnick, D.L., and Stokes, R.A., "The ILLIAC IV Computer", IEEE Trans. on Computers, vol. C-17, No. 8, August 1968, p. 746.

2) Kuck, D.J., "ILLIAC IV Software and Application Programming", ibid, p. 758.

3) Northcote, R.S., "Software Development for the Array Computer ILLIAC IV", Department of Computer Science, University of Illinois at Urbana-Champaign, Report No. 313, March 1969.

Chapter 3
Design Considerations

3.0 Introduction

The purpose of this chapter is to present the spectrum of design considerations which are relevant to the architectural configuration of the Data Management computer system. Additionally, material of tutorial content is included in order to soundly establish an information base against which the design proposed in Chapter 5 may be viewed. Necessarily, more questions are raised than are answered, since many design details fall beyond the scope of the current contract.

3.1 Configuration Considerations

The advantages and disadvantages of a number of possible system configurations will now be discussed. First to be considered is the conventional uniprocessor computer shown in Figure 3.1. Although this configuration is used in the vast majority of computers, it fails to meet the requirements for the space station on a number of different counts. First, in its simple form, the system is incapable of degrading gracefully since there is only one copy of each unit. Redundancy might be added so that components could fail in the processor without degradation; however, similar techniques fail to protect against loss of data from a failed memory. Both the memory and the series of I/O devices may be augmented within limits to increase capacity. However, the processor is not similarly expandable. Finally, since there is only one copy of each element, the system cannot be repaired without interrupting its operation.

To meet graceful degradation and failure tolerance objectives, it is beneficial to configure a system with multiple copies of each of the important units. Figures 3.2 through 3.5 show four possible configurations. Figure 3.2 represents perhaps the most conventional form of multiprocessor computers, characterized by the use of multi-port memory. The system shown uses four-port memories. Each of the ports is connected to a separate data bus which in turn is connected to one of the four driving units in the system, two processors and two I/O controllers. Since the switch components are distributed among the modules, it is straightforward to confine the effects of a switch failure to the locality of the containing module. If the number of memories in the system is at least one greater than the number required to contain all of the necessary information, then this system is capable of graceful degradation. If through software techniques a sequence of snapshots of memory contents is taken to provide recovery from memory failure, and if a time history of input and output messages were main-

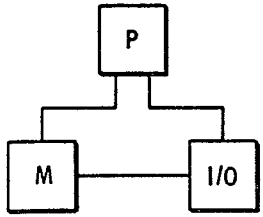


Figure 3.1 Simplex Configuration

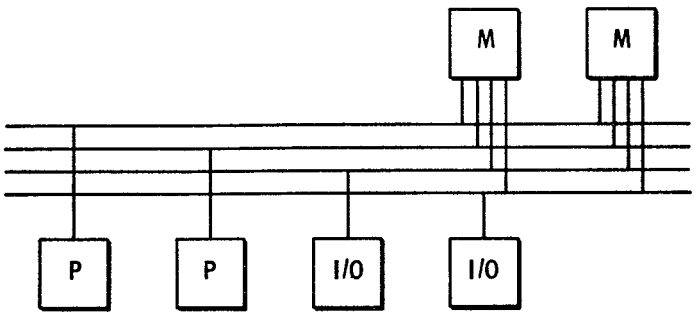


Figure 3.2 Multiprocessor with Multi-port Memories

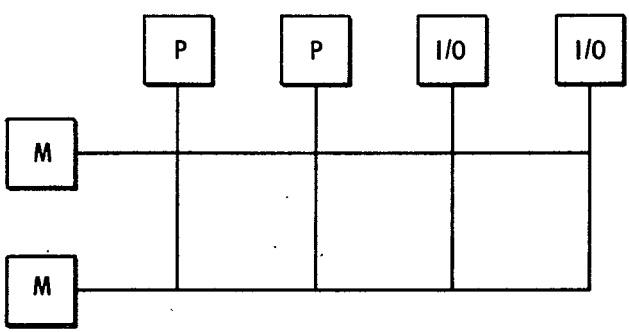


Figure 3.3 Multiprocessor with Crossbar Switch

tained, a software package could be prepared through which memory failures might be tolerated. Should a processor fail, its loss does not disable the other processor nor either of the memories from continued operation. Should an I/O controller fail, it similarly does not prevent operation of the other. The system is modularly expandable or contractable to an extent. That is, more memories can be added with connections for each of the devices present in the system. The addition of processors or I/O controllers, however, requires additional ports on each memory. To enable expansion, the small version of the system could be configured with a number of excess memory ports, sufficient to contain connections for the largest version of the system desired. Such a system would then be expandable, with the upper limit determined by the number of ports on the memory.

The system shown in Figure 3.3 is quite similar in configuration to that of Figure 3.2. The essential difference between the two configurations is that the switching between paths to memory is done in a switch, rather than being built into the memory as in the Figure 3.2 configuration. This system has essentially the same characteristics as the previous one, since memories and processors may be added. It does not have the limitation, however, imposed by a fixed number of ports on a memory. Rather, the limitation comes in the mechanization of the crossbar switch itself. If the switch were built in a modular style, so that additional components of the switch required to support additional elements could be added at the time the extra units were added, then the system would be as expandable as desired. Attainment of necessary reliability in the crossbar switch itself is one of the most difficult design jobs in this system.

The configuration shown in Figure 3.4 differs from those of Figures 3.2 and 3.3 in that communication between memories and processors takes place over common data buses. Since each unit connected to a bus contains logic for recognizing commands to itself, in principle the system can be expanded by merely attaching additional modules to the bus. However, the capacity of the bus itself is the dominant potential bottleneck in this configuration, since the bus can carry no more than one message at any time. However, if this bus is a partially parallel bus (for example, one byte wide) and if sufficiently sophisticated technology is used to allow a high bit rate, then the bus may be made sufficiently powerful to permit substantial system growth. One further distinction is shown in Figure 3.4 which is not necessarily peculiar to this configuration; namely, the use of separate memories for program and data. It is perhaps most appropriate in Figure 3.4, since the bus traffic capacity potentially represents the upper limit on system capacity. If, as in many machines, execution of each instruction requires one instruction fetch and one data fetch, then the provision of a second bus either doubles the system capacity or halves the bit rate requirement for each bus. This system

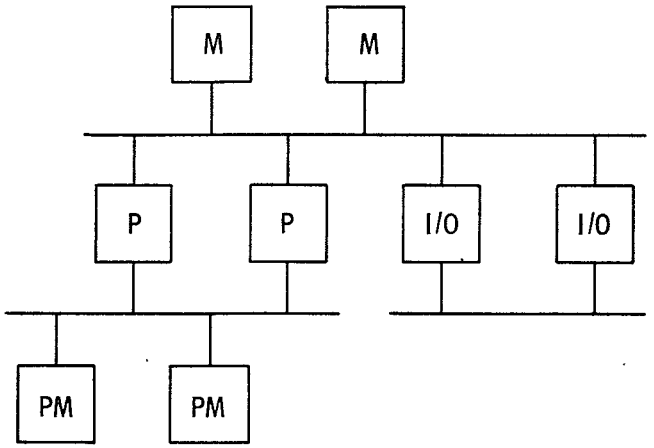


Figure 3.4 Multiprocessor with Common Buses

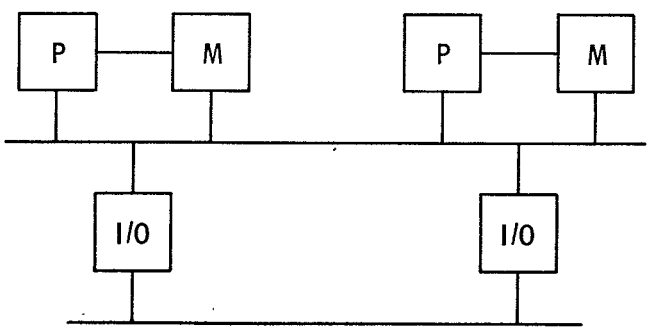


Figure 3.5
Multiprocessor with Common Buses
and
Preferred Memory Paths

is gracefully degradable, since memories and processors may fail without causing the system to be down so long as they do not in any way tie up the bus. However, the bus itself must be infallible since no other communication path is provided.

The configuration shown in Figure 3.5 represents a compromise between a multiprocessor configuration and a multicomputer configuration. This system resembles Figure 3.4 in that a common data bus allows any processor to communicate with any memory. However, the system also represents Figure 3.2 since multi-port memories are used. In this system the number of ports on the memory is exactly two, giving each memory a preferred access path from one processor. Tradeoffs are possible in this configuration between providing small memories attached to processors or relatively larger ones. A small memory would be used as a scratch-pad, whereas a larger memory could contain a substantial fraction of the total system memory and would contain resident programs and data. This system degrades as gracefully as Figure 3.4, since no program's execution depends upon availability of the preferred path between the processor and the memory. However, the existence of this path is intended to greatly reduce the traffic load on the common data bus, which enables the system to grow to a substantially larger configuration before reaching the upper limit of bus capacity.

In each of the systems it must be emphasized that a requirement exists to prevent loss of information. No system can be gracefully degradable if required information is destroyed or lost because of memory failure. The implication of this requirement is that either the memories must be composed of multiple units within each module so that there is a sufficiently high probability that not both copies will fail at once and therefore no information will be lost, or that the system software provides snapshots of data so that loss of memory does not cause loss of data. In the latter case it is necessary to supply an extra copy of memory which is not otherwise used until a failure occurs. After a failure, data present in the failed memory at the last snapshot is loaded into the fresh copy. The implementation of such a recovery technique would impose serious complexity on the operating system and applications programs, since it would be necessary to update the data from the failed memory to the time-state which existed at the time of failure. This complexity is a strong stimulus for rendering data-loss extremely improbable, by using multiple copies of memory within each memory module, or other satisfactory means.

3.2 Trade-off Considerations

Once the configuration has been selected, or in connection with the selection, it will be necessary to make a number of decisions with respect to the major components. The following discussion is intended to illuminate some of the trade-offs.

3.2.1 Processors

a) Should the processors all be alike? Clearly, if an early decision is made to include provisions for unlike processors, the system should operate well even if only like processors are used. Perhaps the converse is true as well, but it seems desirable to consider whether the system requirements tend to indicate that a mixture of processor-types would be advantageous. Several types of processors that might be considered are:

   1) "Standard" units with conventional general-purpose instruction sets;

   2) units which perform floating-point arithmetic substantially more efficiently than the standard units;

   3) units which perform bit-manipulation operations more efficiently;

   4) units especially suited for list-processing operations;

   5) special-purpose units for performing executive or other high-duty-cycle operations;

   6) array processors.

   The advantages of unlike processors are apparent; an offsetting disadvantage is the departure from uniformity, which complicates the graceful degradation property, the repair and spare requirements, and the scheduling software. One approach which appears feasible is the use of alterable microprogram memories in the processors, so that processors could assume any of the identities described above (except, perhaps, the last) by loading the appropriate microprogram.

b) How should processor error-detection be implemented? On one side of the trade-off is the conceptually simple and fool-proof checking scheme in which two or more copies of the processor unit perform identical programs simultaneously and compare their outputs. If it can be assumed that no event which causes an error affects more than one copy, this technique will catch every error. Further, since the processor and its checker are identical, the same spare can be used for both. The alternative is the incorporation of checking circuitry within the processor. Although this

is a substantial complication to the design of the unit, it presumably requires less logical components than does an additional copy of the unit. However, parity or residue checking within the unit cannot detect every error, and a study of the expected probabilities of error-causing events will be necessary in order to determine whether internal checking and the inherent reliability of the unit are sufficiently trustworthy. An additional consideration regarding internal checking is that new module failure modes are introduced: namely, failures in the checking components themselves. The failure mode which causes continuous indication of no error is particularly insidious.

c) Should "scratchpad" or other memory which is locally accessible to the processor be provided? The use of scratchpad storage can be beneficial both as a means of reducing access time to data used in computations, and in removal of traffic from the main communication lines in the system. The latter point is especially significant if a configuration like Figure 3.4 or 3.5 is used. Consider Figure 3.5: if the size of local memory is larger than necessary for the data of the task currently being processed, it becomes possible to use the additional space for the current program, or to assign data or programs to be resident (permanently located) there. If data or program for a task were resident in some memory unit, it would clearly be desirable to execute that task in the associated processor, although that assignment would only increase efficiency, and not be mandatory. However, there would be additional executive overhead introduced as a result of the processor-preference criterion for scheduling; for example, the executive must prevent the occurrence of a queue of tasks waiting for a preferred processor when other processors are idle. Furthermore, whether or not the loss of equality between units by virtue of resident-assignments is favorable or unfavorable to the reliability and recovery strategy requires careful consideration.

3.2.2 Memories

a) Should memory for program and data be separate? Typically, execution of an average instruction requires one instruction fetch and one operand fetch. Separate program and data memories lend themselves to separate bussing, as shown in Figure 3.4, which reduces congestion. Additionally, the fetching of programs from secondary storage, if required, may be done without adding traffic to either bus if an appropriate channel is provided. However, such savings are achieved only by addition of hardware. Loss of flexibility should be avoided by allowing program and data to be in the opposite memory type when convenient.

b) Should memory be paged? While certainly no substitute for adequate memory capacity, paging can be used to increase the effectiveness of the physical memory present. However,

additional software and hardware are required to implement such a system. Even so, use of some form of core-multiplexing appears desirable at this time, in view of the extent to which the computational load predicted for the system resembles the time-sharing load of present commercial systems. Implementation questions such as page or fragment size, number and type of associative registers for rapid access, and software strategy are of immediate significance.

c) How should storage protection be implemented? The wide variety of program and data sources, the desire for on-board program preparation and checkout, and the requirements for high system reliability make storage protection appear mandatory. To some extent, paging would provide such protection, since contents of pages not assigned to a task are not even addressable by the task.

Still further protection is desired, however. No task should be allowed to modify its own instructions by writing into its own program area; further, it would be beneficial to implement array-limit protection, so that no writing addressed to an array would be beyond that array's bounds. Additionally, some subset of the storage protection mechanism should be available to tasks for their own use.

d) Should memory addresses be interleaved among modules? Interleaving is frequently used to permit concurrent multiple memory accesses, to reduce the number of memory conflicts between processors and I/O controllers. In a multiprocessor configuration, concurrent execution of the same program by two processors when no interleaving is provided might cause each to take twice as long to finish as the no-conflict case would have taken. Two-way interleaving would alleviate this problem. However, if one of the two halves failed, everything in memory would be affected. Because recovery from loss of data is so difficult, the design of the memory must make any such loss extremely improbable. If that objective is achieved, interleaving appears desirable.

e) Should certain instructions be physically implemented in the memory rather than the processor? The goal of this would be to reduce the traffic in the communication system of the computer. List-search instructions, for example, might be conducted wholly within a memory module with virtually no use of a data bus. The same is true of the intra-module multiple-word transfers. However, the limited use of such instructions, compared with the estimated cost of the implementation, including logic to deal with encountering a module-boundary, seems to indicate that it would be undesirable. If memory interleaving were used, that would cinch it.

f) How can content-reliability best be achieved? This is a three-fold question; it involves how to make the loss of contents from a given unit improbable, how to detect it if it

should happen, and what to do about it when it is detected. The first is a design-for-reliability question beyond the scope of the architectural consideration. The second and third may or may not be connected, depending on whether, for example, redundant copies of memory are operated to provide error detection and data-backup. Many types of checking codes are capable of error detection in memory operations and are readily implemented. Error correction can also be achieved, although at the expense of additional check-bits and logic. However, address selection errors such as no-word, wrong-word, or multiple-word are not correctible by such means, and unless these can be made sufficiently unlikely, use of extra copies may be the only choice.

g) How much read-only memory should be included in the system? The Apollo on-board computer method of placing all programs in fixed memory is clearly not feasible for the next generation of long-lifetime applications. This is true for two reasons: first, too much on-line memory would be required, and second, ROM is too inflexible (it was in Apollo, too). However, read-only microprograms are frequently used, as are system bootstrap memories for initial loading. The priority of this question is probably rather low.

3.2.3 Communication Paths

a) How many paths should there be? The figures and preceding text have portrayed these tradeoffs.

b) What should each path's width be? Obviously, the path width should be related to the traffic expected, to prevent log jams. The expected traffic is a function of processor and memory capabilities, problem characteristics, and problem mix. The latter two must be expected to still be fairly uncertain at the time this decision must be made, and an approach must be adopted which is quite conservative. The history of the growth of planned function and the extension of the life of the system must be taken fully into consideration.

c) Should control and addressing signals have separate paths or be multiplexed with data? This is both a traffic and a reliability question. That separation of control and data signals reduces traffic on the data bus is obvious; whether it is advantageous to have a separate path to control or be controlled by switching in the event of failures must be investigated.

d) How can communication reliability best be achieved? Questions discussed previously regarding redundancy and checking also apply to the communication net. So-called "transmission" coding can be used to check for and even correct

errors. However, since a viable communications net must always exist even if processor and memory modules fail, the problem is a severe one.

3.3 System Organization for Reliability

3.3.1 Introduction

Reliability, failure tolerance, and graceful degradation are related requirements which must be considered in combination. Reliability is concerned with the probability of failure in the system equipment. It is associated with a time interval, and is either estimated theoretically using a mathematical model, or determined empirically by observation. Failure tolerance is the capability of the system to continue operation after a failure has occurred, whereas graceful degradation implies a gradual reduction of system capability when failures occur. Ideally the system design should:

a) Minimize the probability of equipment failures.

b) Continue full operation even if failures do occur.

Because current technology does not permit simplex construction of a computer having the required life without maintenance, some form of repair or replacement is mandatory. The next subsections describe systems which obtain actual or effective replacements from fixed and open-ended spare pools.

3.3.1.1 Closed System

Consider M to be the number of modules of a given type in the system, and L to be the number of those which must be active to provide adequate capacity. In a closed system, failed modules are not replaced, and M must be about four times L to achieve a .99 reliability over five years, given a failure rate of .0002 per hour per unit[10]. In such a system, an estimate of system reliability can be obtained by forming the product of the reliabilities of all modules, with the switching functions allocated to other modules, or to a hypothetical switch module, as appropriate. Each factor is a function of the number of gates, the failure rate of gates per unit time, and time. Using the data provided in reference 10, and considering an example of a processor of 10,000 gates with the failure rate of 10^{-8} per hour, it is clear that to achieve a reliability of .9 or greater over five years in a closed system requires either a significant decrease in failure rates, or redundancy. However, a triply-redundant system is less reliable than a single unit after 7/10 of the mean life of an individual unit. Thus, there are substantial reasons why a closed system will not provide a ten year life with

currently available gate reliability.

3.3.1.2 Open System

The space system will be an open system, in which failed modules are replaced. In this case, the overall reliability of the system involves the probability of the occurrence of a second failure before the first is repaired. In this sense, reliability encompasses more than equipment reliability. In an open system, the probability of mission success (PMS) is a measure of adequacy which is somewhat less dependent on reliability than it is in the closed system. This is because the PMS is defined as the probability that the computer system will perform at or above specified operational levels, which are time-dependent. Thus, a dip in performance capability may be harmless to mission success if the performance required during the dip happens to be low. The PMS is a function of the time to repair and replace modules as well as the reliability of the modules. A reliability model which includes repair statistics will be generated for purposes of analysis. However, it is intuitively evident that lower module MTBF's can be tolerated in an open system for a given reliability than in a closed system.

3.3.2 Graceful Degradation

The term "graceful degradation" refers to the diminished relative operational capabilities of the system after one or more permanent failures have occurred. The ability of the system to continue its function after a failure has occurred in an element or module is usually achieved by using either redundant on-line modules or off-line modules which can be connected to the system after the error has been identified.

Assuming that there are "critical" functions being performed by the DMCS and that it must be operational for ten years, there are two basic approaches:

3.3.2.1 Standby/Active Approach

Given that L active modules of a given type are required for processing, and that M of these are provided in the system, then (M-L) of them may be kept on standby status, awaiting activation after failures among the L modules. Thus, the system can survive (M-L) failures without any degradation of system performance. In order that certain functions of the system may be continued when more than (M-L) modules have failed, levels of operational priority must be established, since less processing can be done by the system as more modules fail. Figure 3.6 shows an example.

In this case, we see that the system continues to perform 100% of its functions as long as L modules are available. When

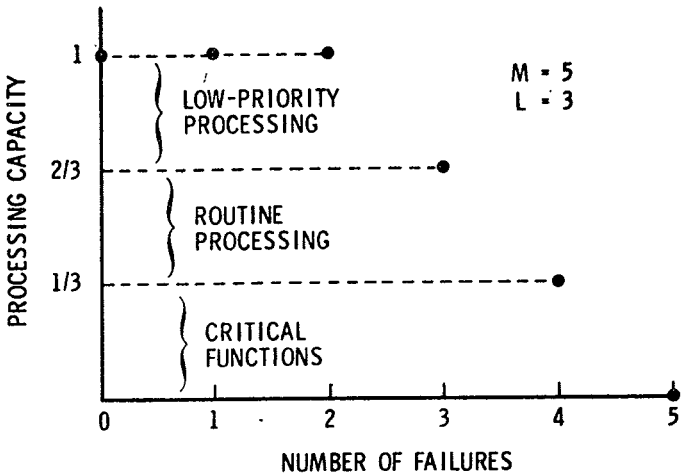


Figure 3.6 Graceful Degradation

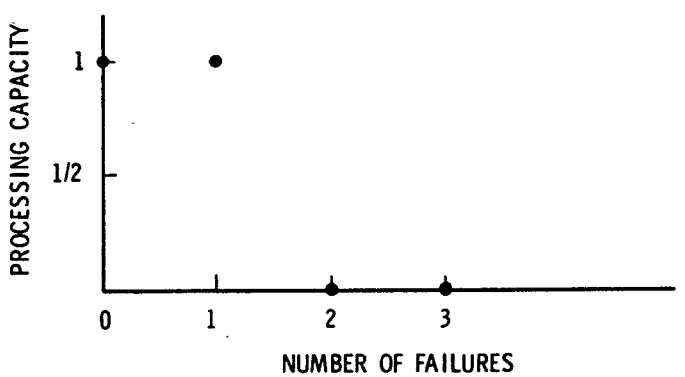


Figure 3.7 Degradation of a Triply-redundant System

less than L modules are available, selected functions are eliminated. When all but one module has failed, the system continues to perform only those functions considered essential, and ignores others. The number of levels selected and the functions associated with each are arbitrary and will define the degraded modes of the system. Probabilities can be predicted using the model for each level. The overall goal for system reliability is critical to the selection of the number of modules. For example, if L were the maximum number of modules simultaneously required, then it is probable that less than this number will suffice for much of the time. If this were taken into account in the probability model, then M could be smaller, since the joint probability of the occurrence of the (M-L+1)st failure at a time when L modules are actually required is less than the probability of that failure alone.

3.3.2.2 Full Redundancy Approach

In a full redundancy approach, the system has only one level of operational degradation; it will either be operational with 100% capability, or inoperative. For example, a triply-redundant system will provide 100% processing capability until the second failure occurs, at which point it must stop. This mechanization may be adequate in an open system. If the probability of the occurrence of a second failure before the first module has been repaired is sufficiently small that the PMS goal is achieved, then this approach is adequate. Note that a multi-mode redundant system with voting could be used to increase the number of successive failures that the system could tolerate prior to repair.

3.3.2.3 Comment on Approaches

a) In both approaches, the system design must provide a "fail safe" mode if there is any appreciable probability that the set of failures which have occurred prevents the system from continuing. When this occurs:

   1) It must recognize the situation, and communicate it to the crew and other computers.

   2) All non-critical functions must be terminated, and the system automatically put into a dormant mode, receptive to direction from the operator. Perhaps hardware functions will be required which automatically set status and control bits, so that the system is left in a truly safe configuration. Apollo G&N computer experience has shown that this simple-sounding procedure can in fact be extraordinarily difficult to implement because of the time-varying nature of what "safe" really implies.

b) Other factors for consideration with respect to these approaches are complexity of software, error detection capability, etc.

c) It may also be consistent to require configuration of either type mode as a function of mission phase. That is, during critical phases, the system might operate in a redundant mode, but in the active/standby mode otherwise.

3.4 Elementary Reliability Based on Queueing Theory

3.4.1 Introduction

The purpose of this section is to illuminate some of the trade-offs which must be resolved in designing a configuration intended to allow "graceful degradation" by inclusion of more than one unit of each kind. It is not claimed that the curves included in this document have direct applicability to any particular design; rather, they are intended to be roughly characteristic of several different design-concepts, and provide an intuitive feeling for the relative benefits provided by the concept itself and by varying the redundancy within a given concept.

Throughout this section, failures are assumed to be random, with exponential distribution. This assumption is made because it is expected to be roughly correct, but also because it is mathematically easy to use. It is realized, of course, that non-random failures and random failures of other characteristics may be of significance. However, they are completely ignored herein.

3.4.2 Review of Basic Theory

If we denote the probability of survival (continued successful operation) of an element at time t by s, (t), the exponential distribution assumption may be portrayed as

$$s_1(t) = e^{-\lambda t}$$

where λ is the failure rate, or average number of failures of that kind of component per unit of time. Another way of describing this failure characteristic is to say that the probability that a unit which is operational at time t will fail by the time t+dt is λdt, and therefore independent of t itself.

Consider now a system repair station or maintenance man which is capable of working on a single problem at a time, and whose probability of completion of a repair which is in process at time t by time t+dt is $\lambda' dt$. If the population of units potentially requiring repair is large, the probabilities that none,

one, two (and so on) units are failed and not yet repaired are described respectively, by

$$\frac{dP_0}{dt} = -A\ P_0 + S\ P_1$$

$$\frac{dP_1}{dt} = A\ P_0 - (S+A)\ P_1 + S\ P_2$$

$$\frac{dP_i}{dt} = A\ P_{i-1} - (S+A)\ P_i + S\ P_{i+1}$$

where $i \geq 1$, and A and S represent the rates of arrival (failure) and service (repair) which characterize the system and the repair facility. The assumed initial conditions for this set is that all units are initially working: $P_0(0) = 1$, and $P_i(0) = 0$ for $i \geq 1$.

Given values for A and S, these equations could be integrated numerically to obtain at least the P_i's for small i's. However, we postpone discussion of time-dependent solutions temporarily, and instead consider the steady state solution. In the steady state, all the dP_i/dt are zero, and the solutions may be obtained step-by-step in terms of P_0 by starting at the top.

$$P_i = P_0\ (A/S)^i \qquad i \geq 0$$

Since the P_i's are mutually exclusive and cover all cases, P_0 may be found from

$$\Sigma\ P_i = 1$$

Thus, $P_0 = 1 - A/S$, and

$$P_i = (1 - A/S)\ (A/S)^i \qquad i \geq 0$$

Under these conditions, the average number of units not operational may be readily computed from

$$m = \Sigma\ i\ P_i = A/(S-A)$$

Several cases are illustrated below:

A/S:	0.1	0.5	0.9
P_0:	0.9	0.5	0.1
P_1:	0.09	0.25	0.09
P_2:	0.009	0.125	0.81
m:	0.111	1.0	9.0

Notice that as the average failure rate approaches the average repair rate capacity, the number of units awaiting repair grows quite rapidly.

3.4.3 Application to a Finite Population

In the preceding analysis, the assumption of a large population permitted treatment of A and S as constants which were independent of the state of the system. Consider now a small population representative, say, of the number of processor elements in a multiprocessor computer system. If we ignore the possibility that the pressure of a long waiting line at the repair facility will have an effect on the repair rate (one way or the other), S may still be considered constant. However, the probability that one of the operational units fails in a specified interval is strongly dependent on the number which are already in the failed state: indeed, if none are working, the probability that one more fails is zero. Thus,

$$\frac{dP_0}{dt} = - A_0 P_0 + S P_i$$

$$\frac{dP_i}{dt} = A_{i-1} P_{i-1} - (A_i + S) P_i + S P_{i+1} \qquad 1 \leq i < N$$

$$\frac{dP_N}{dt} = A_{N-1} P_{N-1} - S P_N$$

where A_i is the system failure rate when i units are already in the repair queue.

If A is defined to be the failure rate of an individual unit (previously, A was the collective failure rate of a large ensemble of units), then $A_i = (N - i)A$ and

$$\frac{dP_0}{dt} = -N A P_0 + S P_1$$

$$\frac{dP_i}{dt} = (N-i+1)A P_{i-1} - [(N-i)A + S]P_i + S P_{i+1} \qquad 1 \leq i < N$$

$$\frac{dP_N}{dt} = A P_{N-1} - S P_N$$

The steady-state solution of this set is given by

$$P_0 = \left\{ \sum_{i=0}^{N} \frac{N!}{(N-i)!} (A/S)^i \right\}^{-1}$$

$$P_i = \frac{N!}{(N-i)!} P_0 (A/S)^i \qquad 1 \leq i \leq N$$

A few illustrative plots of the solutions of the differential equations are shown in Figure 3.8. The scales are non-dimensional; time is expressed in units of At, and the repair rate in terms of S/A. The curves indicate what might be expected intuitively, in that as time increases, the system quickly leaves the initial all-operational state $P_0 = 1$, while the probabilities of n units in the repair queue increase. The P's for larger n's increase more slowly than for smaller n's, since the first transitions into the n state come from the n-1 state.

Figure 3.9 shows a plot of the steady-state solution for a number of cases, but presented in a different light. Again with S/A as a parameter, the probability that all units are in the failed state is plotted against number of units in the system. Notice that the probability scale is logarithmic, to allow display of the wide range of values involved.

Figures 3.10 and 3.11 display the probabilities that less than 3 and less than 5 units are operational in the steady state, as a function of number of units in the system. The motivation for this form of display is that the situation whose probability is plotted is the one in which a system requiring at least 3 or 5 units for full performance of its functions is below that level. Again, several intuitive expectations are borne out. First, notice the curvature of the plots for lower values of S/A, which represent cases where the average unit repair rate is little greater than the unit failure rate. This curvature represents a tendency towards a horizontal asymptote, and reflects the fact that when the repair facility is slow, addition of units causes little reliability improvement. This is because the failed units wait in the repair queue so long, that the added

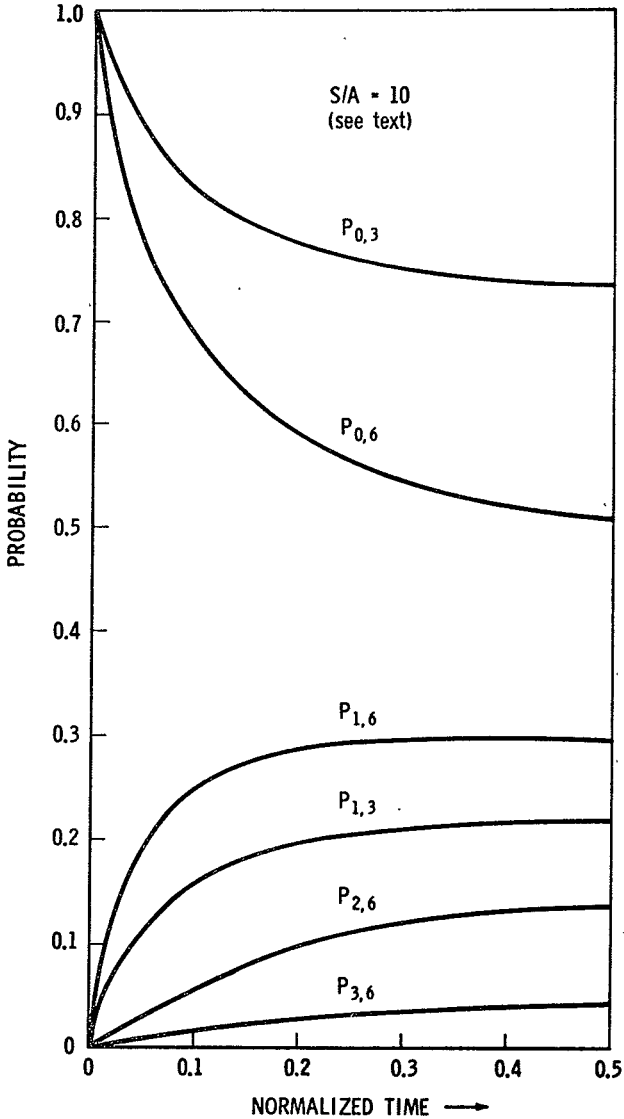


Figure 3.8

Probability that m of n Units are Awaiting Repair Vs. Normalized Time

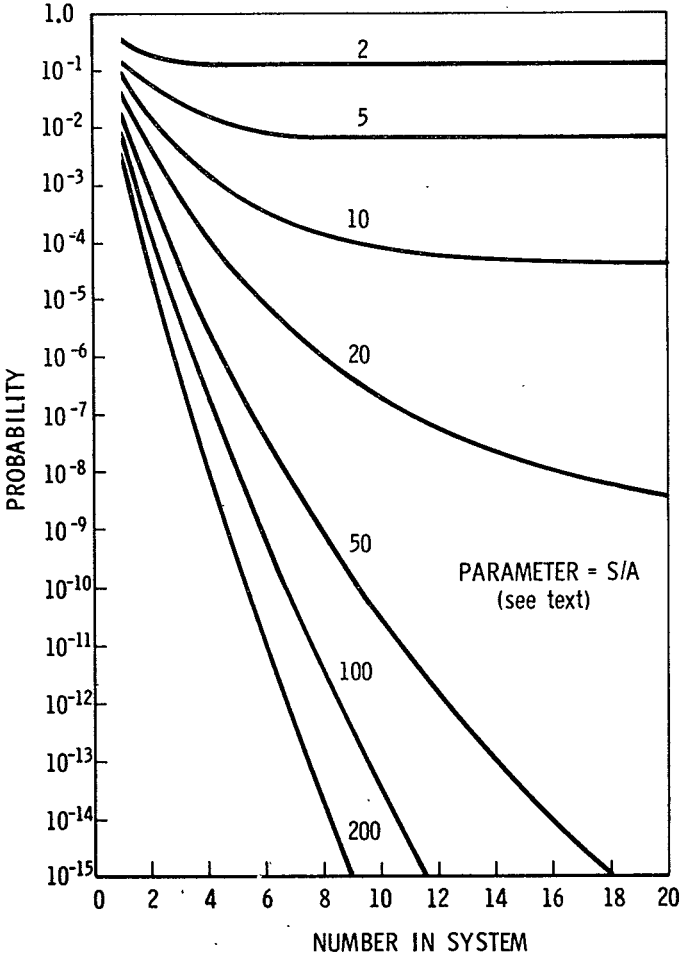


Figure 3.9

Probability that no Units are Operational vs. Number of Units in the System

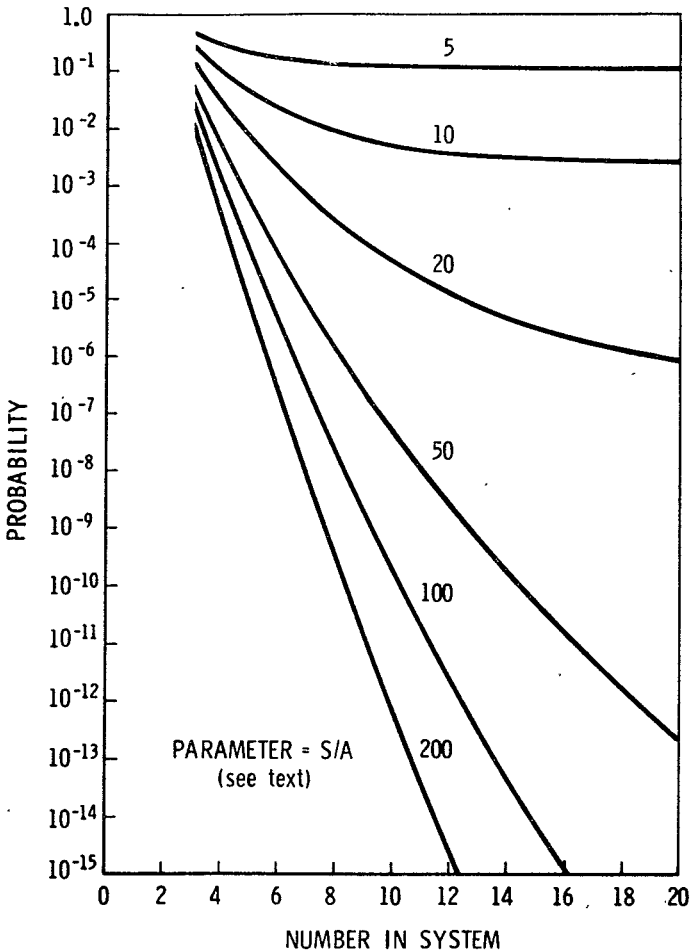


Figure 3.10

Probability that Less Than Three Units will be Operational vs. Number of Units in the System

units are very likely to fail themselves before the others get fixed.

The second intuitive belief is that the number of spares, on a per-required-unit basis, should diminish as the number of required units increases. That is, if a given reliability is achieved when three units are required and five units (1.67 x 3) are provided, then better reliability should result if five units are required and 1.67 x 5 or 8.33 units are provided. For example, in Figure 3.10 it is seen that the probability that less than three units are operational when five are in the system and S/A = 100 is 0.00006; however, Figure 3.11 shows that for the same S/A, the chances that less than five of 8.33 are operational are about ten times lower. Unfortunately, some of the happiness that this brings to the system designer is lost when the difficulties of adding a third of a unit to the system are considered.

3.4.4 No On-board Repairs

Finally, an alternate concept for system maintenance is considered. Suppose that all the units of a type are either hooked into the system or that on-board spares can be swapped with failed units so quickly that it is as though they had been in the system. Further suppose that failed units are not repaired on board, but rather the replacements are brought up for those units on the next periodic shuttle flight. Then, if the shuttle flight period is T, the state of the system tends to diminish with time over the interval, but is restored to perfect condition every T units of time. The differential equations for one interval of this case are simply

$$\frac{dP_0}{dt} = -A_0 P_0$$

$$\frac{dP_i}{dt} = A_{i-1} P_{i-1} - A_i P_i \qquad 1 \leq i < N$$

$$\frac{dP_N}{dt} = A_{N-1} P_{N-1}$$

for $0 \leq t < T$, with $P_0(0) = 1$, and $P_i(0) = 0$ when $1 \leq i \leq N$. If we change variables so that henceforth t represents what was t/T before, and if the former relation $A_i = (N-i)A$ is substituted, the equations become

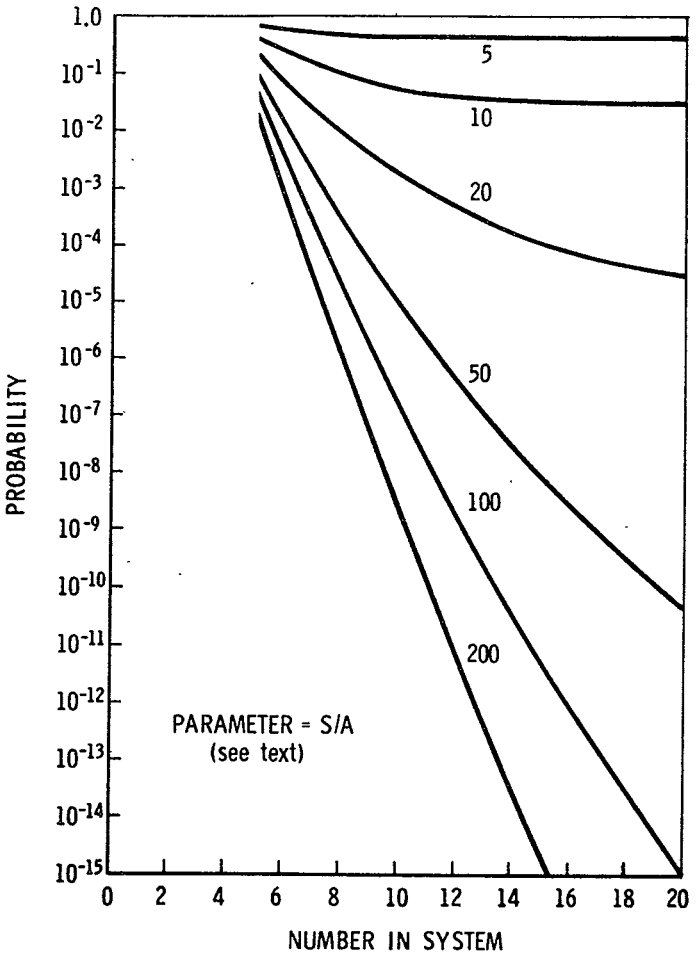


Figure 3.11

Probability that Less Than Five Units will be Operational Vs. Number of Units in the System

$$\frac{dP_0}{dt} = [-N\ P_0]\ AT$$

$$\frac{dP_i}{dt} = [(N-i+1)\ P_{i-1} - (N-i)P_i]\ AT \qquad 1 \leq i < N$$

$$\frac{dP_N}{dt} = [P_{N-1}]\ AT$$

for $0 \leq t < 1$. This system never really reaches a steady state in the zero derivative sense, but to avoid the nuisance of the time dependence, Figure 3.12 has been constructed from the average values of the P's over an interval, with 1/AT as the parameter.

3.4.5 Conclusion

The figures in this section seem to show that reliabilities of multi-unit systems which require only a fraction of the total number of units to be working can be made quite high. However, it must be stressed that these results are based on models of the failure process. It is important for the reader to realize, as he probably has already, that no accounting has been made for the fact that to increase the number of units in a system requires more than just more units; unfortunately, more connections, switches, and other components must be added as well, and often it is the unreliability of these that dominates the system performance.

Another oversimplified consideration is that the failure rates of units are independent of each other and of the level of their own activity. It is well known that failure rates of many kinds of electronics increase with temperature, and decrease when power is off. Failures induced by power switching have also been ignored, but are potentially significant.

The basic conclusion, however, seems clear: if a means can be found for constructing a system so that redundant units can be utilized without introducing appreciable unreliability via their own inclusion, the system reliability can be made almost arbitrarily high. The design proposed in Chapter 5 is believed to possess these attributes.

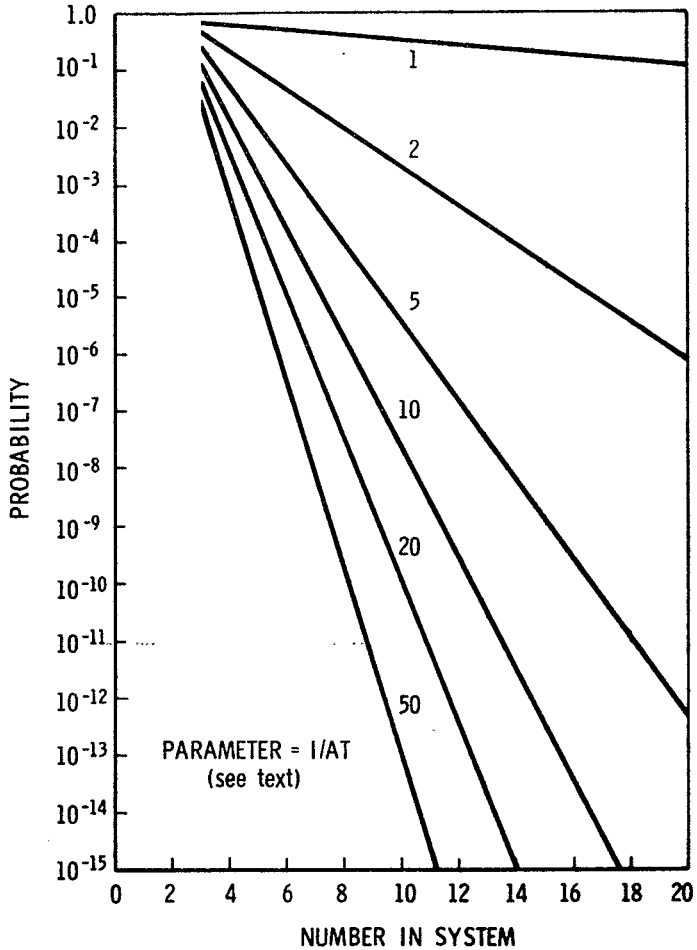


Figure 3.12
Probability That Less Than Three Units Will Be Operational
Vs. Number of Units in the System

3.5 Segmentation and Paging

3.5.1 History

The handling of the problem of allocation of memory in computers has evolved over the years in response to the changing usage of the computer as a system. In the early systems, only one job ran at a time, and the entire computer resources were available for use by that job. Core allocation as such did not exist; the individual programmer was simply responsible for insuring that his program would fit in the storage available. If this was insufficient, he was required to break his program up into pieces which would fit, and to plan their sequential execution. This process is referred to as overlaying.

The next significant change to storage management occurred when multi-programming was introduced. In this case, more than one job could simultaneously be active in the system, and a decision had to be made regarding allocation of space to each. Time-shared systems introduced an even greater dimension, since response-time seen by the user at a terminal became an important parameter in the system operation. The first concept introduced to solve the storage allocation problem was known as relocation. At the time when a segment of a program was to be executed, it was preprocessed by a program called a relocating loader which would customize the program for that instance of execution by changing the addresses in the program to correspond to the physical memory locations from which the program would be executed. Subsequently, hardware was added to the processor to aid this problem, typically in the form of relocation registers. This removes some of the problem of relocation, since it was performed dynamically in the hardware. However, the binding of several program segments together to run was still required since each of the segments was written as though it was to be executed in the low numbered addresses of memory. As the number of users occupying resources of the computer at a given time has grown, the responsibility for core allocation among them has been awarded to a supervisory program. As time-response has become as important as processor efficiency, more exotic address mapping hardware has been added to the processor.

The manifestation of the fundamental storage allocation problem is storage fragmentation, or fractionation of free storage into multiple, relatively small, pieces. This phenomenon is partially caused by the general inability to anticipate storage requirements even over seemingly short time intervals, but it is somewhat unavoidable without hardware aid. An illustration may prove helpful: suppose there are ten units of storage, numbered 0-9. Suppose also that the allocation algorithm awards the lowest-numbered smallest piece of available space which is big enough to satisfy the request. Consider the following sequence, where the number is the space involved, and R or F indicate whether the transaction is a request or

finish (return of space): 2R, 5R, 2F, 4R. It is seen that the 4R request cannot be satisfied even though five units are available, because of the fragmentation of free space.

Two methods used to circumvent this problem are described in the following sections; a survey of systems using these methods is given in Appendix A.

3.5.2 Paging

Paging is a form of address mapping which was first utilized in the Ferranti Atlas Computer. It was introduced to help solve two problems which are an inherent part of time-shared computer usage:

1) It is desirable to execute programs which are not wholly loaded into memory or which will not even fit in the available memory space.

2) It is necessary to remove programs from memory and replace them with ones more currently required, and later to restore them, without substantial storage allocation overhead.

The notion of paging is simple enough: a level of indirect addressing is added to cause logical addresses issued by a program to be translated into physical addresses corresponding to the current location of the block of the users program or data referred to. The list which describes the translation is referred to as a page table, and is addressed implicitly by the processor when needed.

Paging has permitted the user to write his program as though it were to execute in a large virtual memory, the correspondence between the virtual address space of his program and the physical address space computer being accomplished at execution time. In execution, the reference by a program to a page not currently in memory causes a missing-page interruption. The supervisory program then initiates a fetch of the desired page, meanwhile giving control to another process awaiting execution. This strategy is referred to as demand-paging. The combination of poor strategy for selecting pages to be replaced in core plus overambitious attempts to crowd too many users into a given memory have caused some notable performance disasters when paged systems have become overloaded.

One of the important characteristics of paging is that it is invisible to the programmer. This means the programmer need not be aware of the fact that he has other than the virtual memory which he envisions when his program is prepared. This can be an advantage since it frees him from problems of storage allocation. It, however, can also be a disadvantage, since it prevents him from being able to influence memory allocation. Since pages are usually fixed length blocks, it is difficult for him to arrange

the contents of the blocks so that the pages are meaningful logical units of program or data. As a result, sometimes large fractions of pages are filled with information not immediately relevant to that which is being currently used.

Hardware aid to the paging process takes several forms. First, the processor may incorporate high speed memory for storage of part or all of the translation address words, to reduce the time penalty caused by the extra indirection in addressing. Second, the hardware can control the settings of bits to indicate those pages which have been referred to and those pages which have been written into since a given time, in order that the page switching software can determine whether it is necessary to write a page out to secondary storage when its space is preempted to make room for another page. If the page has not been modified, it need not be written out, since a copy already exists on the secondary storage device. Third, the hardware might (but normally does not) keep an ordered list of page references so that the software could determine with a minimum of overhead which page was least recently used when space for a new page was required. Fourth, the hardware can readily implement storage protection by providing bits in the translation address word which indicate the page is a read-only page, an execute-only page, or a free read and write page. It should be noted that pages not known to a process by virtue of being included in its page table are protected automatically, since they are simply not addressable by the process and therefore, are completely safe from over-writing.

That paging is an effective means of memory allocation depends upon a characteristic of programs in execution for short periods of time: namely, that the accessing of words in program and data is not uniformly random, but rather is confined to a small subset with high probability[6]. Thus, if the period of execution of a program is brief, for example, one time slice, much of the program and data will not be referred to during the interval, and therefore need not occupy space in memory. The extent to which this hypothesis is true in a given application can profoundly affect the success or performance of a given implementation. As a result, many articles have appeared in the literature describing different paging measurements and strategies.

3.5.3 Segmentation

Segmentation is a generalization of the virtual memory concept through the provision of a series of independent virtual memories. Each one of the virtual memories may be considered to contain exactly one segment, so that any segment may grow or shrink without affecting other segments. Further, segments not in use during execution of a program need not be physically present in memory. That is, segments need be loaded only when referred to. This is useful since the largest unit of program

which must be bound together, as described previously, is the segment. Since the virtual addressing within each segment may begin at location zero, and since these addresses are dynamically, not physically, relocated during execution, segments may readily be shared between processes without elaborate additional mechanism.

It is seen that some of the characteristics of segmentation overlap those of paging. Indeed, if the typical size of a segment was of the order of the size of the page, paging as such would not be useful as an additional characteristic of the hardware. On the other hand, if segments are often substantially larger than the page size, paging is useful.

The use of segmentation is often referred to as two-dimensional addressing, since the address of an item in a segment is specified by a segment number and a relative location within the segment. Therefore, unlike paging, segmentation is a logical division of address space which is completely visible to the programmer, and need not be inherently related to the problem of memory allocation.

Two of the most recent systems to be based on segmentation are the MIT Multics system using the G.E. 645 computer[5] and the Burroughs 6500/7500 computer system[3]. In Multics, both segmentation and paging are provided. In the Burroughs system, only higher order language is used for program preparation, and the structure of these languages is used to inherently define rather small segments. Thus, paging is not required, except for large data arrays. In both systems, segmentation is used to segregate read-only procedures or programs from alterable data. Additionally, both systems rely on segmentation to achieve orderly sharing of programs and data among processes. In both, segment descriptor words are used for location translation of segment addresses. Segment length information is also contained in the descriptor word, and is used to validate addresses as they are issued. Burroughs uses this feature to the extent that each array is defined to be a segment, so that illegal subscripting can be discovered by hardware, eliminating software overhead for this.

3.5.4 Paging Studies

3.5.4.1 Theoretical Consideration of Paging

The material presented in this section is based on the content of reference 7.

Let m be the probability that a page-fetch is demanded by a particular program. This is a function of the size of the program, the number of programs resident in memory, and the memory size. Let T be the traverse time of a page-fetch, which is the sum of T_a, the access time of the page on the secondary storage

device, and T_t, the time to transfer the block of words comprising the page. Let time be measured in microseconds, making the time-unit roughly comparable to the instruction execution time of the processor. If a page is defined to be 1000 words, the following table indicates typical speeds:

Device	T_a	T_t	T
IC or film	.1	10^2	10^2
Core	1.0	10^3	10^3
Bulk Core	10	10^4	10^4
Fast Drum	10^4	10^3	10^4
Moving-arm Disc	10^5	10^3	10^5

Suppose a job is running for an interval of time over which the missing page probability, m, is approximately constant. We wish to compute the paging efficiency, which is defined to be:

$$e(m) = \frac{RT}{RT + \text{Page wait time}}$$

$$\text{Page wait time} = m \times RT \times T$$

$$e(m) = \frac{RT}{RT + m\ RT\ T} = \frac{1}{1 + m\ T}$$

Figure 3.13 is a plot of e versus m for various values of T. It illustrates dramatically the need for small values of the product m T.

At first glance, it appears that a low paging efficiency for one job could be compensated for by running enough jobs simultaneously to keep the processor occupied. This falls short on two grounds:

1) The objective of paging (neglecting interactive users' response times) is to multiplex core to keep the processor efficiently loaded. But if each job has only a low duty cycle, then more core will be needed, not less.

2) The paging device, often a drum, will saturate and become the limiting item. In running a set of 10% efficiency jobs, the drum may continually be fetching pages, and the processor will be idle while all jobs are waiting for pages.

The only real answer lies in providing enough pages so that each job can run with a reasonably high efficiency.

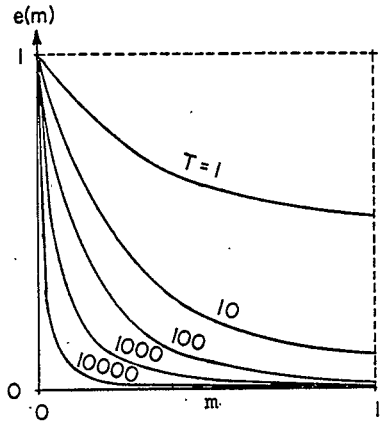


Figure 3.13
Paging Efficiency

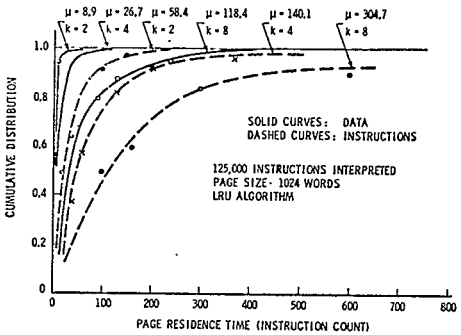


Figure 3.14
Page Residency Distribution

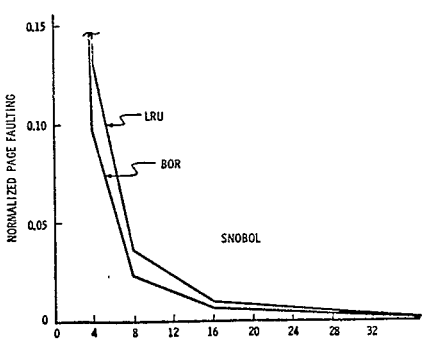


Figure 3.15
Comparison of LRU
and BOR Algorithms

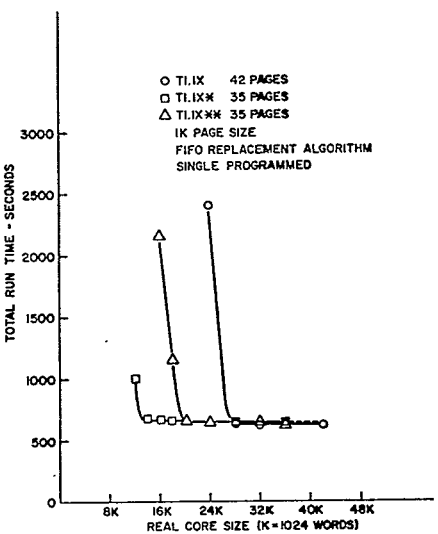


Figure 3.16
Effects of real core size
Tl-Matrix inversion(100x100)

3.5.4.2 Experimental Studies

Results of several studies are presented without extensive comment. First are some measurements by Varian and Coffman[4]. Figure 3.14 is a plot of what is called "page residence time" distribution. Actually, it is the distribution of the time intervals between consecutive page faults for a particular problem under differing allocations of number of pages. The test problem is a SNOBOL compiler for the IBM 360, which consisted of 15 instruction pages and 22 data pages, each of 4096 bytes. The results for data and instruction pages are given separately. The quantity labeled μ is the mean time between page faults; k is the number of pages allocated. As can be seen, for 8 out of 15 instruction pages, the mean time between page faults is only 300 instructions; for 8 out of 22 data pages, the mean is 120 instructions.

Figure 3.15 shows the overall picture for the same problem. The ordinate, labeled "normalized page faulting", is the number of page faults divided by the total number of instructions that were executed. The abscissa, labeled k, is again the maximum number of pages that this problem was allowed. As mentioned above, the size of the problem was 37 pages. The scaling is not good at large k's, but even for a k of 24 to 30 the amount of paging is not negligible. (LRU = Least Recently Used, BOR = Belady Optimum Replacement, two strategies for page replacement.)

Another study was done at the IBM Thomas J. Watson Research Center[1], using a specially modified 7044. In these cases, core size was varied, and the actual time to complete the problem was measured. The secondary storage device had a traverse time that was 25,000 times the core memory cycle time. Three jobs were selected to be run.

1) A FORTRAN job that inverts a 100 by 100 matrix.

2) A FORTRAN job that does data correlation using a fair quantity of input information.

3) A sorting job that sorted 10,000 10-word items.

Figure 3.16 shows the run time as a function of the alloted core space for the first job. The original problem is given by the points marked by the circles. The triangles and square represent versions of the program that were reprogrammed to reflect the paging environment.

Figure 3.17 illustrates the second problem results. This time the triangles stand for the initial program and the circles for the improved version.

Figure 3.18 gives the times for the sort problem. The solid rectangle is the basic program and the others were succes-

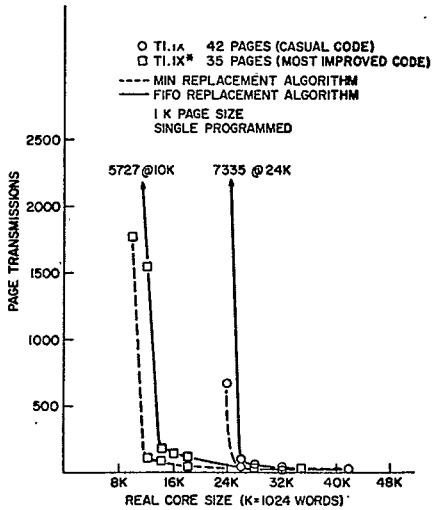


Figure 3.17

Effects of page replacement algorithm
T1-Matrix inversion (100x100)

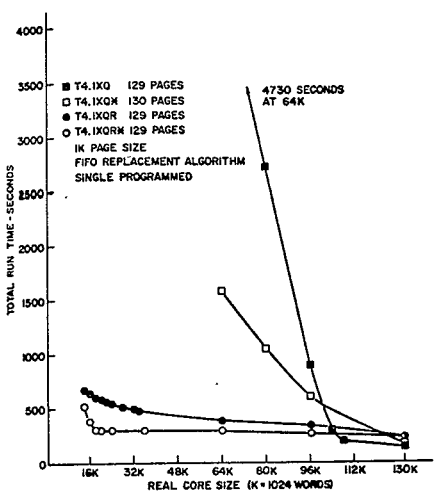


Figure 3.18

Effects of real core size
T4-Sorting (10,000 10-word items)

sive improvements. The rather dramatic savings in core needed were achieved by using the large file of data (100 pages) in small sub-files at the cost of additional processor time. Less processor time is required when the whole file can be randomly addressed and a list threaded through it.

The thrust of the conclusions was that acceptable performance can be realized if programming techniques are used which recognize the paging environment. If allowances are made by the programmer for the need of the paging mechanism to shrink the allotted size of memory available for his program, then it is possible to produce programs that will run efficiently under paging.

Two factors should be mentioned which might influence the extrapolation of the results:

1) The compilers and programs were taken from an IBM 7044, a second generation computer. Techniques and program layout methods have changed since then: re-entrant code, pure procedures, involved interaction with large operating systems, etc.

2) The jobs all had one characteristic in common: they were larger than 32K in their natural form, and had all been reworked to make them fit into a 32K configuration by overlay techniques. (Perhaps the breaks in the run-time of the FORTRAN jobs near 32K are no coincidence.)

3.6 Processor Interrupts

3.6.1 Introduction

The purpose of processor interruption is to alert a processor to the occurrence of an event, while eliminating the necessity for repetitive testing under program control. Each interruption causes that processor to record sufficient information to resume the interrupted process at a later time, and then to begin execution of instructions at a location corresponding to the particular interruption. Hardware interrupt features are an integral part of the design of most computer systems. Their implementation, coupled with the executive scheduling and dispatch functions in the software, provide the overall control structure for the configuration.

Multiprocessing systems introduce an extra dimension for design consideration. Such questions as "which processor should be interrupted" or "should one processor service all interrupts" appear in addition to the questions in simple processor system design such as "should there be a priority structure for interrupts", "what are the hardware functions for interrupts", and "what software control of interruption-disabling is possible".

The purpose of this section is to review interrupt features of several existing systems.

3.6.2 History and Background

Historically, the early computers operated in a serial manner, in that initiation of each action had to await completion of the previous action. It was frequently necessary to afford careful consideration to execution timing in order to synchronize data transfers between memory and the peripheral devices.

Later, computers such as the IBM 709 enabled interleaved instruction execution and I/O operation. These systems included one or more input/output channels capable of executing sequences of I/O commands themselves without the participation of the processor. When processor aid did become necessary, the channel was able to trigger an interruption. This system had the advantage of permitting program execution and I/O to operate concurrently. The interruption of the running program upon completion of an I/O operation was accomplished by causing an involuntary transfer of control to a predetermined memory location, at which an interrupt service routine began. Multiple and time-shared I/O channels were subsequently introduced, which increased the possibile multiplicity of the I/O operations and the complexity of the interrupt servicing.

3.6.3 Single-Level Interrupt

In a single-level interrupt structure, the processor is in one of two modes, the "normal" mode or the "interrupt" mode. When the processor is operating in the normal mode and a condition occurs which, by design, requires an interrupt, the processor is placed in the interrupt mode, and control is transferred to a predetermined location in memory. The concept of single level is simply that while the processor is in the interrupt mode, it may not again be interrupted. That is, recognition of further interrupts is postponed until the processor leaves the interrupt mode. This inhibiting of interrupts requires the hardware to be designed such that no data or interrupts are lost.

When the interrupt occurs, certain information must be preserved so that the interrupted program may later be resumed. This information includes the processor state, plus the set of machine registers which could be over-written during the execution of the interrupt servicing routine.

3.6.4 Multi-Level Interrupt

The multi-level interrupt structure typically assigns each interrupt or class of interrupts to a "priority level". The

hardware is designed to allow interrupts assigned to a given level to interrupt those of a lower level. Otherwise, the basic concepts are similar to the single-level interrupt structure.

An unusual system of this type, the Litton 304, uses 64 levels, and delegates a significant amount of the executive control to this hardware. Each process (or program) as well as each interrupt is assigned a level, and the system essentially never leaves the interrupt mode.

3.7 Stacks

3.7.1 General Description

A stack, according to Knuth[11], "is a linear list for which all insertions and deletions (usually all accesses) are made at one end of the list". Stacks, which have proved to be important in many computer applications, especially recursive procedures, have been called by many other names. Among them are "push down lists", "last in, first out (LIFO) lists", "cellars", "nesting stores", and even "yoyo lists".

When an item is put onto the top of the stack the process is called "pushing down"; to take an item off the top is to "pop up". The bottom of the stack is the oldest word in it, and hence the least accessible item. When a stack is pushed down to accept an additional item, the words in the stack in memory are not physically moved from one location to the next. Instead, a variable, called the stack pointer, contains the address of the location in memory of the top of the stack, and is merely incremented when an item is added to the stack. Thus the items in the stack appear to be pushed down because their location is farther away from the location pointed at by the stack pointer.

Usually there are limits or bounds on the memory space that the stack may occupy. If the stack size violates these bounds, the condition is called stack "overflow" or "underflow".

3.7.2 Examples of Stack Implementations

A stack system has been implemented strictly through software for many computers. However, several computer manufacturers have recognized the utility of stacks, and have implemented hardware and instructions which facilitate stacking mechanisms. Some of the computers which have implemented stacks are the following:

1) SDS Sigma 7[14]. The Sigma 7 resembles the IBM 360 in its data format and special registers. It includes byte, halfword, word, and double-word data handling instructions. How-

ever, its instruction set is quite different, and it has a stack capability. The instructions are as follows: Push word, Pull word, Push multiple word, and Modify stack pointer. The effective address portion of each of these instructions points at a location which contains the "stack pointer double word" (SPD). The operation can be seen by consideration of the "Push word" instruction which increments the stack pointer, and then takes a word from an accumulator and stores it in the location pointed at by the new contents of the stack pointer. Stack limit data in the SPD is used to validate the operation before it is actually performed.

2) DEC PDP-10[8]. The stacking capabilities of the PDP-10 are similar to the Sigma 7. The two common instructions are PUSH and POP. The PUSH instruction is like that of the Sigma 7, except that the sources of data and stack pointer are reversed; i.e., an accumulator contains the stack pointer, and the effective address specifies the location of the data to be transferred to the stack.

   Two more instructions that are useful are PUSHJ, which causes a transfer to the addressed subroutine, leaving return information in the stack, and a return counterpart, POPJ. In detail, PUSHJ increments the stack pointer accumulator, pushes the program counter and flag information into the stack, and jumps to the location specified by the effective address. POPJ provides the means to return. This pair of instructions is a useful mechanism for handling nested or recursive subroutines.

3) Burroughs 5500, 6500 and 7500[2,3]. The Burroughs implementation of the stack is no half-way measure. Rather than offering a stack as an optional feature which may or may not be used by the programmer, Burroughs has incorporated the stack into the fundamental architecture of their computer. The pair of registers that are sources for operands and destinations for results (effectively, the accumulators) are logically considered by the hardware to be the top of a stack. Arithmetic operations such as ADD take their inputs from the top of the stack and leave their results on the top of the stack. Other instructions are provided to move operands from memory to the top of the stack, or store data from the stack to memory. The top of the stack, then, is the heart of all calculations and the stack itself shrinks or grows as the computational sequence indicates. This utter dependence upon a stack rather than multipurpose accumulators seems to be unique to Burroughs.

4) DEC PDP-11[9]. The PDP-11 contains eight "general registers", two of which are dedicated to specific functions and are implicitly addressed by certain operations. One of these is the program counter; the other is the stack pointer (SP). The addressing modes of the machine autoincrementing (increment register after use) and autodecrementing (decrement register before use) which facilitate the use of other re-

gisters as stack pointers also. The interrupt sequence in the processor and the subroutine call and return instructions store appropriate information in the stack by use of SP, so that priority interrupt and nested or recursive subroutine implementation is quite direct. Hardware-aided checks on stack limits are almost non-existent; a single fixed address in memory is treated as the stack upper limit, and a trap occurs if a stack controlled by SP exceed this boundary.

3.8 Microprogramming

The term microprogramming was introduced in a paper by Wilkes in 1951[15]. The intent was to introduce "a systematic alternative to the usual somewhat ad hoc procedure of designing digital computers". The traditional technique was to specify only the inputs and the outcome of each individual instruction and leave the details of the implementation to the logic designer. Wilkes pointed out that the execution of an instruction involved a sequence of information transfers, and compared these individual steps to the execution of individual instructions in a program. Each step can be considered a microinstruction; the complete set then constitutes a microprogram.

Microprograms usually reside in a device distinct from the users' memory, called the control storage. Although control stores have generally been read-only memories, several computers have recently been developed with read-write control stores. This opens the door to intriguing possibilities, such as dynamic variation of the instruction set the particular computer might possess. Use has been made of this capability to build emulators for existing computers and to write diagnostic routines which perform machine checks at a more basic level than could be accomplished with ordinary instructions.

The significant advantage that microprogramming offers can be described by one keyword: flexibility. Profitable uses of this flexibility include the following:

1) Many times it becomes apparent during the software-writing effort that certain additional instructions would make them more useful, but it is too late in the design cycle to incorporate these changes. Microprogramming allows the redesign of an instruction set (within limits) long after the basic hardware intself has been frozen. This situation arose in 1968, when IBM redefined the floating point arithmetic on all 360 models. For the most part, this change was accomplished by merely rewriting and debugging the appropriate microprograms. To have made a corresponding change at the hardware level would have been enormously more expensive[13].

2) Similarly, microprogramming offers the opportunity to provide instructions and special features tailor-made to a customer's

unique needs long after the system has been specified, with no hardware modifications. The addition of special purpose hardware and peripherals, not supported in the original design, to an already designed system has historically been an awkward and expensive task. Microprogramming offers real potential for solutions to this problem.

3) As an extension to the capability to emulate older computers, microprogramming may prove quite valuable as a test-bed for the development of future computers. Emulators may be written for proposed machines and measurements conducted to offer empirical evidence of the design efficiency.

Examples of contemporary microprogrammed computers include the following[13]:

1) IBM 360/25. The 25 has a writable control store for which load decks that make it look like either a 360 or a 1400 series computer are supplied. Since the control store is generally writable, other emulators could be produced for the 25.

2) IBM 360/85. The 85 has two microprogram control stores. One is read-only, and contains the 360 emulator. The other is generally writable, and supports the 7094 emulator as well as basic machine diagnostic routines.

3) Standard Computer Corporation's IC Series. Standard has introduced a whole series of computers which are microprogrammed. The earlier ones were designed especially to simulate the IBM 7090/7094 and 7040/7044. Newer machines in the Standard line offer the possibility of emulating a number of different common computers of several manufacturers. The IC7000 is particularly slanted towards the time-sharing market.

Enthusiasts have raved over the possibilities and seemingly unlimited potentials of general microprogramming. They envision the ability to manufacture a computer that could emulate the characteristics of every commonly used computer. By the relatively minor amount of programming necessary to construct the microprograms (1800 instructions in the Standard Model 9)[12], they can cash in on the huge investment in time and money already spent to produce operating software. The technique would all but eliminate the gigantic reprogramming costs of switching over to a new generation computer. The following quotes illustrate some of the claims that one proponent of microprogramming is zealously putting forth: The "life of existing program libraries will be extended to infinity". "Vintage software, massaged and made workable through frequent use and long study, can now be employed as required without locking the user in or out." "We are rapidly approaching the time when all programs will run on all machines."[12]

References for Chapter 3

1. Brawn, B.S., and Gustavson, F.G., "Program Behavior in a Paging Environment", Proc. FJCC, 1968. Vol. 33, pp. 1019-1032.

2. Burroughs Corporation, "A Narrative Description of the B5500 Disk File Master Control Program". May 1965.

3. Burroughs Corporation, B6500/B7500 Characteristics Manual. Sept. 1968.

4. Coffman, E.G., and Varian, L.C., "Further Experimental Data on the Behavior of Programs in a Paging Environment", CACM 11,7. July 1968. pp. 471-474.

5. Corbato, F.J., "A Paging Experiment with the Multics System", MIT Memo MAC-M-384. July 8, 1968.

6. Denning, P.J., "The Working Set Model for Program Behavior", CACM 11,5. May 1968. pp. 323-333.

7. Denning, P.J., "Thrashing: Its Causes and Prevention", Proc. FJCC. 1968. Vol. 33, pp. 915-922.

8. Digital Equipment Corporation PDP-10 Reference Handbook. 1969.

9. Digital Equipment Corporation PDP-11 Handbook. 1969.

10. Erwin, F.D., and Bersoff, E., "Modular Computer Architecture Strategy for Long Term Missions", Proc. FJCC, 1969.

11. Knuth, D.E., The Art of Computer Programming, Vol. 1, Addison-Wesley. 1968.

12. Rokoczi, L.L., "The Computer-within-a-Computer - Fourth Generation Concept". Computer Group News (March 1969), pp. 14-20.

13. Rosin, R.F., "Contemporary Concepts of Microprogramming and Emulation". Comp Surveys 1,4 (Dec. 1969), pp. 197-212.

14. SDS Sigma 7 Computer Reference Manual. 90-09-50F. Dec. 1968.

15. Wilkes, M.V., "The Best Way to Design an Automatic Calculating Machine". Manchester University Computer Conference, 1951. pp. 16-21.

Chapter 4

System Design Guidelines and Constraints

4.0 Introduction

This chapter presents the major guidelines and constraints influencing the architecture of the computer for the space station data management system. The scope of this contract did not include any detailed analysis or generation of system design requirements; however, it is a meaningless exercise to attempt to configure a large-scale system without at least order-of-magnitude estimates of requirements. Therefore, this chapter is intended to summarize existing space station DMS guidelines, and preliminary DMS design requirements being developed by the Space Station Phase B contractors.

We observe that at present, neither the design criteria for the computer system nor the operational and performance requirements it must satisfy are well defined. Terms such as high reliability, on-line reconfiguration, graceful degradation, and configuration flexibility are being used rather loosely as characteristics of the computer system. These terms all have broad scope in meaning, and their exact interpretation with respect to the space station has a direct effect on the architecture of the computer system. In addition, the processing requirements of the on-board computer system in terms of "what it must do" have so far only been grossly estimated, based upon preliminary functional analyses. These obviously are not adequate to finalize performance requirements or sizing of the computer system.

However, these requirements, guidelines and constraints have been used in planning and designing the organization of the computer system presented in Chapter 5. It is therefore useful to restate them, and to interpret them where necessary. The information is presented in the following sections: General Space Station Subsystem Requirements, Performance Requirements, Physical Requirements and Reliability.

4.1 General Space Station Subsystem Requirements

With reference to the "Statement of Work Space Station Program Definition (Phase B)", 14 April 1969, the space station will be designed for a minimum of ten years of operational life, and each of the subsystems will be designed with large margins and provisions for in-flight maintenance, repair, and replacement. In addition, the station will be designed to take advantage of technological advances in subsystems which occur after it becomes operational.

The Intermetrics interpretation of this follows:

a) The data management computer system (DMCS) should be designed to have an operational life of at least ten years.

b) The DMCS should be designed assuming that in-flight repair or replacement of failed modules will be performed, and that the supply of spares may be replenished via the shuttle.

c) The DMCS should be designed to take advantage of technological advances which occur during its life. This implies that the initial system is not a "closed system", or one in which all equipment is available from the beginning, within the initial configuration.

   It should be noted that expansion of requirements over the operational life of a system has always been underestimated. It therefore appears reasonable to establish a guideline that the computer system be designed with an expansion safety factor of about four. That is, the capacity of the design, if not the initial implementation, will accommodate a quadrupling of requirements over the life of the system.

d) It is assumed that the software will be expanded and otherwise modified during the life of the system. It is also assumed that on-board software generation capability is to be provided, plus provision for testing and introducing of new program modules created on-board or on the ground.

e) The system must detect all permanent and transient failures which result in errors. In addition, it must distinguish between permanent and transient errors, identify the modules which contain permanent failures, and recover automatically.

f) The power supply shall be decentralized, and implemented so that no power failure at a modular level can disrupt system operation.

g) The interface of the DMCS with other systems shall be designed with a capacity in excess of the predicted traffic, by about a factor of four.

4.2 Performance Requirements

The major contributions to the load on the data management computer system is estimated to be the processing and control of experiments. Approximately 70-90% of the storage requirements and 50-70% of the speed are expected to be absorbed by programs and data for scheduling, initializing, and controlling experiments, plus data collection and computational services[2]. Therefore, a careful examination of these requirements is necessary to ultimately size the system.

Prior to discussing the storage and processing requirements, a brief overview of the anticipated operational functions is provided.

4.2.1 Functions of the DMCS

At the current stage of the space station program, the design of the major operational functions is not complete to the level of detail necessary to assess their full impact on the computer's sizing. Further, it is our estimate that this will not be fully resolved during phase B of the program.

In general, the DMS computer system will be interfaced with a number of subsystems on board, and will serve as the primary computation facility. Some of its principal interfaces with on-board sensors and subsystems are: Control and Display Subsystem (probably CRT-like devices), an Inertial Subsystem, Digital Communications Subsystem, Rendezvous and Docking Radar Subsystem and other Docking Sensors, Surveillance Radar, Reaction Control Subsystem, Primary Propulsion System, Balancing Subsystem, Power System, Experiment Equipment Interfaces Environment, Thermal Control, and Biomedical Subsystem. It is assumed that the DMCS will send and receive information over the external data bus, and provide the control and processing required by these subsystems.

The following major operational functions will be supported by the DMCS:

a) Primary and Command and Control

One of the prime functions of the DMCS is to drive the displays for command of the space station. The computer will assist the crew in planning and execution of maneuvers, flight decisions, and trajectory control, and will provide other data for flight control of the space station during its mission. This will include functions such as rendezvous and docking.

b) On-Board Checkout

Another function of the DMCS is the periodic checkout of the on-board subsystems to determine whether or not they are operating in an acceptable manner. There are several aspects to on-board checkout: status monitoring, in which test points are checked to determine if any gross faults exist; trend analysis, for predicting faults; and diagnostics to determine malfunction location to provide a basis for reconfiguration actions. In addition, some form of failure correction, calibration, and record keeping are part of these functions.

c) Mission Planning and Operations Scheduling

The computer will assist the crew in performing mission analysis and assessment, and in daily crew scheduling and logistics inventory control.

d) Guidance, Navigation, and Control

The computer will, using its sensor subsystems, maintain knowledge of position and velocity of the station. It will also perform the artificial G stabilization, and attitude control for pointing of earth survey instrumentation.

e) Experiment Command, Control and Data Processing

As stated above, one of the largest tasks of the DMCS is predicted to be the processing of data from various experiment modules. Some of these functions include experiment scheduling, experiment command and control, data collection, data formatting and storage, data reduction and processing, and display interfaces.

There are many experiments planned for the space station over its life. Four of these experiments predicted to have the largest impact on data input to the computer system are: (5)

Advanced Stellar Astronomy

Plasma Physics - Subsatellite

Earth Surveys

Remote Maneuvering Satellite

f) Software Support System

This portion of the DMCS includes the software operating system. For purposes of this organization of functions, it includes utility software required to support other aspects of the computer system.

4.2.2 Summary of Phase B Preliminary Sizing Estimates

4.2.2.1 Storage

Preliminary estimates of the size of and storage requirements for the space station computer system have been made by M-D/IBM, the MSFC Phase B contractor, to be about 300-500K words of operating main storage and $7 \times 10^6 - 4 \times 10^7$ words of bulk storage (36 bit words). Supporting assumptions

for these estimates are included in reference 2.

4.2.2.2 Processing

IBM has estimated the speed required of the computer to be not more than 10^6 equivalent add operations/sec[2]. This is currently above the speed of presently available airborne computer systems; however, we believe this estimate to be low. Data processing centers, in many ways comparable to the DMCS, operate in excess of this figure. In fact, some contemporary ground-based systems are achieving more than 10^7 ops/sec. However, to estimate the speed requirement accurately requires deeper resolution of the functional requirements, which is not possible at this time.

4.2.3 Digital Input Data Rates[5]

A preliminary estimate by NAR of the total input data rate to the DMCS from all experiments suggests that 310×10^9 bits/day is the upper limit. However, with scheduling of the larger data-gathering experiments as proposed by NAR, 90% of all experiment requirements can be achieved within a limit of 180×10^9 bits/day. The computer system will process and compress this data so that only a small percentage of it need be maintained in the files or sent to the ground.

4.3 Physical Requirements

There are no currently existing physical requirements such as power, weight, and size for the DMS computer system. Some figures, however, for existing airborne systems are presented in Appendix A of this report.

4.3.1 Modularity

Another constraint on the system is that it be modular. We have interpreted the meaning of modularity, and present the following general requirements: the system will be composed of a number of modules best defined by their physical characteristics. Each module will be a subunit which is physically self-contained, and which can be replaced without a major disassembly of the entire system. Each is connected to the system at a number of points for power, information (I/O), thermal control, and physical support. Memory, processor, and I/O units are examples of modules.

a) If a module fails permanently, it will be replaced. Each module will be constructed to maximize its reliability, and will include internal redundancy if appropriate.

b) The number of different types of modules in the system will be minimized, to facilitate maintenance and testing.

c) The electrical interface for each module will be simple, with the minimum number of pins necessary to satisfy performance requirements with reliable technology.

d) The interface for each module will be standardized to facilitate expansion, testing, etc.

e) Each module will be designed so that it can be removed and replaced on line without shutting the system down.

f) Logical connection of modules to the system must be under both program and operator control.

4.4 Reliability

Two of the most important factors in the trade-off considerations of the configuration design are flexibility (or expandability) and reliability. To date no complete quantitative statement of a reliability requirement exists.

One important assumption which we have made with respect to reliability is that the computer system is performing some number of "critical" functions, those which directly effect crew safety, during the mission. It is considered that these functions must be performed 100% of the time, with interruptions of no longer than milliseconds for recovery from failures.

4.4.1 Failure Tolerance

Reference 4 defines a failure tolerance requirement for the system which allows no performance degradation after one failure, performance at a reduced level with two failures, and fail safe after the third. This requirement is quoted below:

a) Capability shall be provided for performing critical functions at a nominal level (performance of operations for which the system was designed) with any single component failed or with any portion of the subsystem inactive for maintenance.

b) Capability shall be provided for performing critical functions at a reduced level with any credible combination of two component failures or with any credible combination of a portion of a subsystem inactive for maintenance and failure of a component in the remaining subsystem.

c) Capability shall be provided for performing critical functions at an emergency level (sufficient for survival only) until the affected function can be restored or the

crew returned to earth.

Although the above failure tolerance criteria are frequently referred to as reliability specifications, brief consideration shows that they are not. In colloquial language, the "real" reliability requirement is the attainment of a specified probability that the DMCS will be able to perform needed functions at the time they are needed. Clearly, this involves a combination of failure rate, failure tolerance, and repair rate which provides the specified probability that needed performance capability exists. In particular, if the time to repair a failure is substantially smaller than the time to the next failure, tolerance of more than one failure seems unnecessary. On the other hand, less favorable combinations of failure and repair rates can conceivably require tolerance of more failures in order to meet the goal.

Because of the large cost of the DMCS, and the sensitivity of the cost to redundancy and other "reliability"-aids, a much more carefully thought-out specification for reliability and availability is needed for the DMCS than has historically been put together for airborne and space systems. Only when this specification has been created can relevant decisions be made with respect to failure tolerance, error detection, and recovery characteristics of the computer system.

4.5 Information & Display

4.5.1 General

The design of information and display techniques for the space station DMS must provide a sufficient interface for the crew to operate, control, and communicate with on-board subsystems to accomplish mission objectives. Currently, manned spacecraft are filled with many gauges, meters, controls, and computer-generated data displays which permit significant interaction with the pilot. The concept for future advanced spacecraft will include not only more sophisticated subsystems with more automatic processes, but more autonomous operational tasks, and a wide spectrum of scientific experimentation and research over longer mission intervals.

The on-board display and controls provided must therefore emphasize flexibility for multipurpose use and high reliability, but remain a simple, efficient, man/machine interface. The purpose of this section is to supply an overview of the general information and display concepts, discuss control of a multi-processor computer system which is performing a number of independent tasks, and related problems.

4.5.2 Space Station Information & Display Requirements

It is helpful here to review some of the basic assumptions envisioned for the space station, to serve as a background for discussion of the display interfaces with the multiprocessor system.

4.5.2.1 Displays

A general ground rule will be that electromechanical display devices[3] are to be eliminated. These will be replaced by more flexible electronic displays, such as CRT's or other two-dimensional devices. The state-of-the-art in input and output devices for computer systems will certainly change and improve over the next decade.

4.5.2.2 Interactive System Terminal Developments

Most interactive system terminals today use typewriter-like devices. Their prime advantages are their relatively low price and the use of the hard copy medium, which automatically provides a record of all input and output. Cathode ray tube devices avoid some of the problems of typewriters; they can operate rapidly, and are considerably more flexible in format and editing control. CRT's are gradually becoming more widely available as terminal devices, and over the next few years should be increasingly competitive with typewriter devices. Obstacles to their acceptance include high cost for terminals, lack of hard copy, and communications limitations, which make the rapid data rate necessary to remotely maintain distant displays prohibitively expensive. Prices are coming down slowly, and the continuing influx of reasonably inexpensive keyboard-plus-CRT alphanumeric terminals has accelerated the trend away from paper output devices.

Other techniques are being investigated which will facilitate new methods of dialogue with computers in the future. These include direct use of handwritten input via devices such as the Rand Tablet or Grafacon, and even voice input and output through a set of software and hardware constructions. Interesting demonstrations and papers are being presented on on-line, hand-written input.

Some research work is underway into audio-input systems. LISPER is a limited speech recognition system developed by Bolt, Beranek, and Newman.[1]. LISPER operates within certain limitations. First, there are a hundred items in its vocabulary. Second, the number of speakers is limited, and each must first be trained by the system in closed-loop fashion so that the system recognizes him. Nevertheless, it has been successfully demonstrated.

Audio output is also available today on some ground-based systems, and may have application in the space station program.

Another approach to computer input which has been tried is a so-called "list selection technique". This technique involves lists which are displayed either on a CRT or optical screen, and which may be changed rapidly under computer control. The user composes input into the system by selecting words and phrases from the list. This can be done either with a light pen, or in the case of the CDC Digiscribe, by touching electrically conductive regions on the face of the CRT with the finger. As the user selects phrases from the list, new lists are displayed as required. This approach to computer input takes advantage of the wide bandwidth of this class of displays, and of the human eye, to rapidly convey information to the user. He may then respond manually at a low rate. It is particularly useful for users who are not good typists.

4.5.2.3 Space Station Terminal Complement

However attractive some of the developmental techniques appear, we propose to limit present consideration to equipment which is certain to be available in time. The initial display equipment needs are assumed to be:

a) A computer controlled, multi-purpose display and control unit to be used as the primary man/machine interface. It will consist of:

   1) A CRT-like display console with state-of-the-art refresh rates, illumination, resolution, and buffer memory.

   2) Keyboard input device with alphanumeric and special character keys.

   3) General purpose function keyboard for single action responses, with flexibility to redesignate function/key assignments dynamically.

   4) Light pen or similar device.

b) Hardcopy device such as a line printer.

c) Microfilm viewer with programmed retrieval capability; to contain schematic and other reference or library data.

d) Closed circuit TV monitoring system.

e) Status and control panels.

f) Direct "joy stick" controls.

The design of these subsystems must conform to space station guidelines of low power consumption, high reliability, long life, compact packaging, and modular construction.

The remainder of this section is devoted to a discussion of Item a. (Control and Display Unit) and the associated design problems.

4.5.2.4 Preview of Manned Operations Aboard Space Station

As presently conceived, the space station will have a central command center similar to the bridge or combat information center of a ship. The command center will be manned on a 24-hour basis, and will contain all controls necessary for operation of the station. An experiment control center, not necessarily co-located with the primary command center, will contain displays and controls necessary to operate and monitor the experiments and other functions of the station.

Functions of the command center include:

1) Flight operations scheduling and control of mission events.

2) Vehicle operation; all operations performed with the vehicle: rendezvous, docking, orbit determination, attitude control, etc.

3) Subsystem status, monitoring, control, and configuration control.

4) Operation and control of the DMS computer.

5) Crew scheduling and training.

6) Communications.

Functions of the Experiment Control Center include:

1) Experiment control and planning.

2) Data collection and compression

3) Ground interface

4) Data analysis

Internal communications and procedures will be established for overlap in control of the base with respect to experiments.

4.5.2.5 Display and Control Information Required On-Board

The information required by the crew to control and operate the space station has a variety of characteristics. Alphanumeric data, graphs, and pictorial data (static or even moving) may all be required. Further analysis is required to resolve the optimum level of information to be presented to the operator with respect to each operational function requiring an interface. To resolve this issue requires an interactive process in which automatic and non-automatic computer functions, and crew interfaces required to best perform mission tasks are evaluated. Hardware/software complexity, availability, and cost will constrain the degree of automation and types of displays made available, whereas crew safety and the complexity of the crew's role may require more advanced display techniques.

Since it is premature to identify all displays, some examples are offered of the types of functions to be performed to indicate the scope of display data requirements. As a general observation, the display and control capabilities will include: alphanumeric and graphic outputs from the computer and alphanumeric, special function key, and light pen inputs.

Examples of functions which involve displays are:

a) Control and operation of the DMS system.

b) Control, selection, and data input/output from operational software (rendezvous tracking, maneuvers, docking, instrument pointing, etc.).

c) Station position and situation displays (orbital position and velocity indications, attitude, rotation rates, thrust controls, extra-vehicular module position, communication coverage, mission events and schedule).

d) System and subsystem situation and status displays (configuration of system, health of subsystems, equipment modes).

e) Interactive data requests (file management, data retrieval, graphs).

f) Experiments displays/control (experiments equipment status, data displays, controls).

4.5.3 Preliminary Information and Display Concepts

The on-board computer system, coupled with its software and multipurpose display and control unit, are the basis for the overall man/machine interface in the space station. Although it is premature to attempt a detailed design of the display system required for all DMS subsystems, some general concepts and

problems for that design are identified.

4.5.3.1 Software Environment

One view of the computer system for the space station is that it is a real-time process control system, providing remote time-sharing services to both batch entry and interactive users. It is unique in this sense. A hypothetical example for comparison purposes might be some large time-sharing service (like Multics) which is also operating a power plant and a surveillance radar. The many varied users of this system establish a need for varied types of displays.

To expand on the requirements for displays for the system, a brief organization of the software environment during system operation reveals four types of software:

a) System Control Supervisor

This class of processing is continuously operating, controlling the resources of the system.

b) Continuous Automatic Sequences

This class of processing includes the functions critical to operation of the space base, and operates automatically and continuously. Examples are attitude control, environment control, system failure detection, and status monitoring and recording.

c) Periodic Operating Processing

This class includes processes which are not in continuous operation, but are of greater importance than some other functions when operating. For example, rendezvous operations, fault isolation, maneuvers, control of external crafts, and some experiment control.

d) Batch-type Data

This class of processing includes utilization of the computer by various subsystems on an as-required basis. This includes scientific experiments, data processing, bio-medical processing, data reduction, preventive maintenance, data retrieval, etc.

These classes of programs involve varied display requirements; however, common to all is the operator's ability to initiate programs, provide data input, receive outputs, and generally to control the operation of the system.

4.5.3.2 Control of the Multiprocessor

4.5.3.2.1 Job Control. The operator must be provided the capability to initiate various sequences or programs to perform specific mission functions. This may be accomplished by a job control language, which should be generalized for all types of job requests and execution. The periodic operational programs and terminal-submitted software require this feature.

The job control language would be the primary interface between the user (or operator) and the operating system. The commands given via this language would be interpreted and result in queues set up for processing by the executive. There are at least two types of users:

a) The remote job entry, whereby a user enters jobs at a terminal and expects them to be executed at some future time.

b) The on-line user who needs to interact with the computer to accomplish this job.

On-line users must be provided with other features which enable easy rapid access to selected programs. Minimum actions should be required to select and operate frequently used programs. For example, the method of selecting programs may be via a light pen action. Program names would be presented on the CRT, and the operator would activate a program by pointing with the light pen. Other data required at program selection time could be entered via the keyboard.

4.5.3.2.2 Protection Philosophy. Input or job requests from terminals should generally be accepted by the system only when the system verifies that the user has appropriate access and execution privileges for the requested job. This may be implemented in the system in many ways: through hardware, software, or both. The degree of built-in interlocks for job requests or file-access depends on the criticality of the damage that can be done and the system implementation and overhead cost.

4.5.3.2.3 Data Output and Display Format. Programs must be capable of requesting action from the operator, such as data entry or verification. Again a language must be defined for use with a CRT display which is generalized for many functions. The Apollo G&N System uses a VERB and NOUN approach to identify actions and data between the operator and computer. With its limited DSKY, it provided much flexibility.

Generally, attempts should be made to minimize the amount of coded output. With an alphanumeric display, direct language

communication should be provided. Coded information requires training or memorizing of codes which at best is error prone.

The Apollo DSKY is designed to transmit commands and requests made up of a limited vocabulary of 99 nouns and 99 verbs. To command the computer, the astronaut depresses the verb (operator) key followed by two decimal digits, and enters a noun (operand) in a similar fashion. The enter key is then depressed, and the computer acts on the request. As an example, Verb 16 Noun 20 means display and monitor spacecraft attitude. Verb 16 means "display and monitor" (continuously update); Noun 20 identifies what to display, in this case "spacecraft attitude".

Features such as: DISPLAY (VARIABLE), DISPLAY VARIABLE EVERY N SECS, DISPLAY VARIABLE IF (CONDITION), PLOT (VARIABLE) VS VARIABLE, etc., are all desirable components of an operator's input language.

Data output from the computer to the operator must optionally include the variable name, value, and units (unless standard units are used throughout). The vocabulary of the computer to the operator requesting action must be simplified. The format of the output frame should be standardized. The conversational vocabulary should be easily identified, perhaps by its location on the CRT, or by size or illumination. Variable names and data should be distinctly set off from this vocabulary to eliminate confusion. A standard header should exist at the tope or bottom of the display and include pertinent data about the console's use, time of day, etc.

4.5.3.2.4 Display Routing. A number of display and control consoles, manned by different personnel performing varied functions (some of which overlap), requires a technique for establishing "who gets what displayed". Furthermore, the command console must be identified to the system as such, since it may be the only console allowed to direct certain critical functions.

From a software point of view, the system is communicating with a number of consoles, not people. When a periodic operational software program is requested, the question of where the output should be sent is fairly straighforward: namely, to the console requesting the job. However, when an alarm is detected by one of the automatic or continuous programs, it may be desirable to output this to a number of consoles. Two solutions to this are (1) dedicate consoles physically to receive such data, or (2) provide a technique for assigning displays to consoles. The latter appears distinctly superior.

4.5.3.3 Parallel Processing with Serial Output

A conflict arises when more than one task operating in the computer requires communication with the same operator.

This problem is not unique to multiprocessor systems, but also exists in multiprogrammed systems.

This problem occurred in the Apollo G&N system when a "background" type program, such as rendezvous tracking, detected an alarm condition while the astronaut was using the DSKY to operate a "foreground" program, such as rendezvous targeting. The design of that system (which is probably unacceptable for the space base) was to bring up automatically a priority display over-riding the existing DSKY display, whether the astronaut wanted to see it immediately or not. Furthermore, he had to respond to it before he could continue with the targeting program operation. Fortunately, this did not occur often, but it caused crew dissatisfaction, and in some cases special exceptions had to be implemented via software to work around this problem.

One possible solution to this problem for the space station is to utilize a portion of the CRT screen (or possibly a separate panel) to advise the operator of waiting displays. This portion could be divided into a number of subsections, each with a sufficient area to display meaningful identification information for a waiting display. The content of the information in this subsection could range from simply a presence-indicator for a waiting display with no identification, to a short identifying message which could include its priority, coded content clue, origin of job, and level of importance.

In any event, the operator would be given the choice of (1) ignoring the waiting request, or (2) selecting a particular waiting display using light pen or keyboard. In the latter case, the system might put the previous display and job into a wait state. This would imply that the old display would be placed on the waiting display queue.

4.5.4 Conclusion

The foregoing description of aspects of the information-handling operations required on the space station indicates that a great deal will be demanded of the terminal equipment and the operating system and terminal-support software. Command and control, real-time monitoring, program preparation, data-processing, automatic checkout, and information retrieval operations are readily foreseen. Furthermore, control of several independent processes from the same terminal is likely to be the rule rather than the exception. This combination of anticipated complexity, plus the additional operations which must be expected to develop over the life of the mission, results in a clear requirement for a thoroughly thought-out, generalized terminal and operating-system philosophy and implementation.

References for Chapter 4

1) Bobrow, D. G., Klatt, D. H., "A limited speech recognition system", Proc. FJCC, 1968.

2) IBM, Draft of Space Station DMS Requirements, S.O.W. item no. 2.1.2-b, c, d, e; undated; informally transmitted Jan. 1970.

3) MSC Space Station Task Group, Space Station/Base Technology Program Plan, Nov. 1969.

4) MSC Space Station Task Group, Guidelines and Constraints Document, MSC-00141, Rev. I, 9 Jan. 1970.

5) North American Rockwell, Space Division, Space Station Program Phase B Definition February Progress Review, PDS70-212, 19 Feb. 1970.

Chapter 5

Selected Multiprocessor Design Configuration

5.0 Introduction

The preceding chapters have presented considerations, alternatives, trade-offs, and requirements which influence the design of the system for the space station. With that material as background, it is the purpose of this chapter to present the design created by Intermetrics. This design is the primary result of the study effort.

There are three major factors which tend to cause consideration of a multiprocessor or multicomputer configuration, rather than a simplex system, for an application:

a) The computing capacity required exceeds that attainable from a single processor.

b) The reliability or availability required exceeds that attainable from a single processor.

c) It is desired to be able to incrementally expand the system without overhauling it.

In the space station application, it is clear that the latter two apply, whether or not the first does. Therefore, the design considerations were concentrated on creating a sound multiprocessor or multicomputer configuration.

At the present stage of the space station and space base programs, both the performance and reliability requirements are incompletely formulated. The forecast of performance expansion required over the five or ten year life of the mission is particularly cloudy, and may, in fact, ultimately be based on the capability of the computer configuration adopted, rather than the other way around. It would be foolish to ignore these uncertainties in formulating a design; instead, Intermetrics has developed an architectural organization which allows implementation of a series of compatible configurations with a strikingly wide range of performance.

5.1 Configuration Summary

The basic organization selected is shown in Figure 5.1. The fundamental characteristic of this configuration is its use of a common internal bus, which eliminates the requirement for multiple buses and switching networks. The simplicity of this organization is most attractive; the threat of a potential bottleneck imposed by the common internal bus is its outstanding

drawback. To avoid congestion which would be caused by heavy bus traffic, each processor is provided with a local memory whose architectural characteristics depend on the required performance. At the low-performance end of the spectrum, this memory is used to contain a push-down stack for storage of temporary results. At the high-performance end, the local memory contains a stack, but also acts as a high-speed buffer store similar to the "cache" of the IBM System/360 Models 85 and 195[5,10,11,12].

The bus recommended for each of the configurations is the same. Although this bus is capable of sustaining the processing level of the maximum system, its capability, which is not expensive, is used effectively in the lower performance models. The manner in which this is achieved is clear if one views the M1 memory as a device for reducing the per-instruction bus-use frequency for its processor. At lower performance, M1 contains only a stack, and perhaps a few words of instruction buffering. If the processor speed were increased without modifying M1, the frequency of bus-accesses would increase proportionately. However, the introduction of a cache would effect a traffic reduction. By adjusting the size of the cache (buffer) memory, the average bus-use per instruction can be controlled, so that bus-use per processor can be made fairly uniform, independent of processor speed.

The above considerations reveal that processors of differing performance may be attached to the bus, since they are compatible in every way except performance. (That this is true depends upon the fact that the buffer operation is invisible to the software; this is explained in section 5.2.1 below.) As a result, performance of the system can be changed over a very wide range after it is in operation, merely by adding, removing, or replacing one or more processors.

Two other levels of memory are provided in the organization. The second level, shown as M2 in the figure, assumes the roles played by both high speed main memory and drum storage in most commercial time-sharing or data-processing systems, since it is both sufficiently fast and large. Finally, because the subset of programs at any given time represents only a fraction of the total, a third level (M3) of memory is included for bulk storage. Of course, the amount of M2 and M3 memory in the configuration is incrementally variable, and may be selected or varied to meet system operating requirements.

The system shown in Figure 5.1 represents the Intermetrics belief that a multiprocessor configuration can best meet the requirements of the application. However, the proposed organization is also excellent for a single-processor configuration, should one be required for the mission or for decentralized subsystem testing.

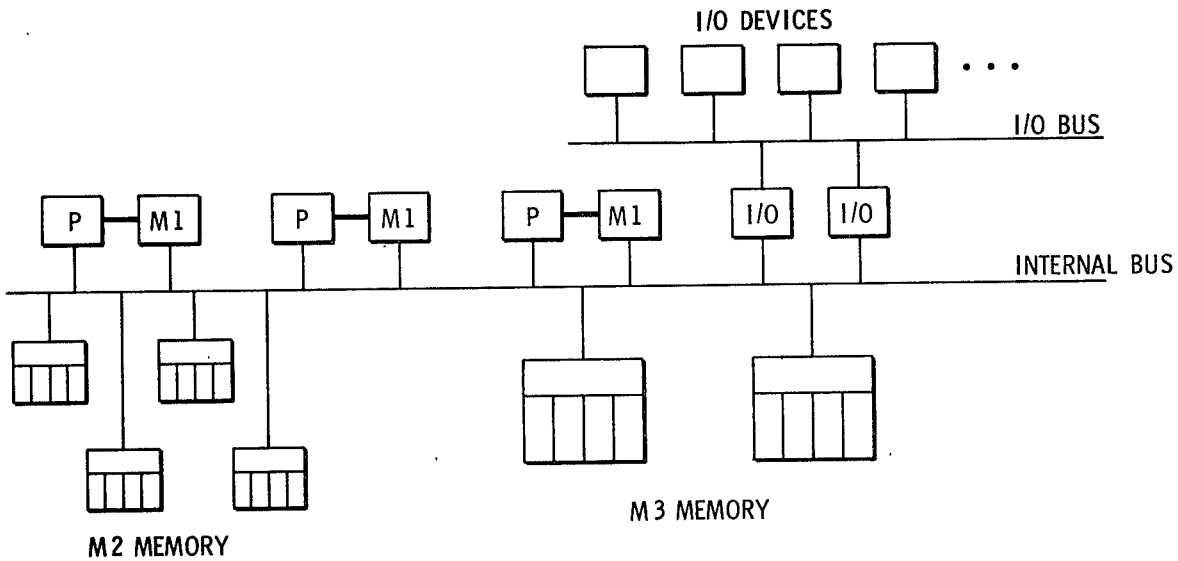


Figure 5.1 Intermetrics Multiprocessor Series

5.2 Buffer Memory

Because of the key role played by the buffer memory in the operation of the high-performance system, it is described first.

5.2.1 Design Rationale

The access time seen by a processor attempting to fetch instructions and data directly influences the maximum operating speed. For a processor to be capable of operation in the 1 to 10 million instructions per second (mips) range, this access time must be from 0.1 to 1 μsec. Additionally, to sustain such a rate, the volume of accessible data must be large, since a sequential series of 10^6 executed instructions normally spans a considerable number of program and data words. The two requirements of speed and capacity conflict, however, since increased capacity at a given speed leads to increased physical size, with attendant signal propagation delay increases (not to mention the cost penalties). Fortunately, it is characteristic of typical programs[1,6] that the accesses to instructions and data tend to be localized, over short time intervals. As a result, a split-level memory can be used to great advantage to provide, at one level, a very high-speed modestly-sized store, and at the other level, large capacity at readily attainable speed.

In operation, the processor issues fetch requests to the buffer memory. If the addressed word is currently contained in the buffer, it is sent to the processor without requiring a fetch from the M2 memory. Otherwise, the buffer initiates a fetch of a group of words from M2, and retains the group for future use.

To avoid performance degradation that would be caused by software overhead, control of the contents of the buffer memory is implemented wholly in the hardware. This results in the desirable characteristic that the presence of the buffer is entirely invisible to both the application programs and the operating system. The relative performance is then a function only of the speed ratio between the average access times for the two memories and the probability that a given access finds the word is not in the buffer. For if there are n accesses in time T, if the average access times seen by the processor are t_m and t_b for main and buffer memories, and if the probability that the accessed word is not in the buffer is p, the word rate W is given by

$$W = \frac{n}{T} = \frac{1}{pt_m + (1-p)t_b}$$

Let $t_m/t_b = R$

then $W\ t_b = \frac{1}{1 + (R-1)p}$

where $W\ t_b$ is the normalized word rate. For R = 10, $W\ t_b$ as a function of p is shown by

p:	0.9	0.5	0.1	0.05	0.01
$W\ t_b$:	0.11	0.18	0.53	0.69	0.92

Thus, if a value of 10 can be maintained for R, and if p can be made as small as 5%, the system will perform at about 70% of the speed of one whose entire memory was as fast as the buffer. Studies performed in connection with the configuration design of the 360/85[12] concluded that a 16 kilobyte buffer is adequate to attain an average miss percentage of 3.2% in the absence of task switching. The main memory of the 85 is 0.5 to 4 megabytes in size, so that buffer capacity represents from 3% to 0.4% of the main memory. IBM estimated that multiprogramming would degrade the miss frequency to about the 5% level. The Model 195 whose buffer memory (cache) is differently organized, and larger, is reported to achieve a 1% miss-rate[7]

5.2.2 Operation Details

With respect to buffer operations, the main or M2 memory may be considered to be composed of a large group of small blocks of, say, eight words. As described above, fetch requests issued by the processor for words currently contained in the buffer cause no M2 memory operation. However, when the addressed word is not currently resident in the buffer, the buffer issues a block-fetch request to M2 over the internal bus. M2 responds by returning the group of eight words, which are retained in the buffer in a location chosen automatically by the hardware, based on a least-recently-used strategy. The word originally requested is forwarded to the processor as soon as it arrives at the buffer. The main memory address of the fetched block is stored in the buffer with the data, to allow the buffer to recognize subsequent fetches from the same block.

Three types of store operations are separately considered. Normal stores will always cause main memory to be updated; if the addressed word is in the buffer of the processor performing the operation, it too will be modified. If not, the buffer content will be left alone. Each other buffer unit which contains the same block will respond to the store message on the bus by marking its corresponding block invalid. Should its processor subsequently address that block, it will experience a

normal buffer-miss condition. The immediate updating of main memory eliminates the need to copy modified blocks from the buffer back to main memory when their space in the buffer is pre-empted, and also means that processes running on other processors, and I/O operations, use current data.

The second type of store occurs when data is placed into main memory by an I/O controller. In this case, since the chances of that data residing in any buffer are small, each buffer will respond to such a message on the bus by simply marking its corresponding block, if any, invalid. Again, should its processor subsequently address that block, it will experience a normal buffer-miss condition.

The third type of store operation may be generically termed multi-process-critical stores. The non-interruptible test-and-then-set instruction mentioned in Chapter 1 is perhaps the best example of this class of operations. There, the explicit intention of the instruction is to provide the mechanism for air-tight interlocking between processes. If this instruction were to follow the operational sequences outlined above, integrity of execution could only occur if access by any other unit to the addressed location in M2 was inhibited until the fetch from M2, testing by the processor, and restoring into M2, was complete. This is readily implemented, since all affected units are linked by the common bus.

Perhaps it is appropriate to mention here that the test-and-then-set instruction is troublesome in another way: a processor encountering a locked lock must either execute a loop, which includes the test, until the test is satisfied, or delay further operation until it receives a signal that the lock has been unlocked. The former places a potentially heavy load on the common bus, which will degrade the speed of the very operations whose completion is awaited; the latter requires provision of a specific lock-clearing instruction whose execution additionally causes delaying processors to re-test the locks they are waiting for. The second approach is superior, and readily implemented.

5.2.3 Characteristics

Although more detailed examination of both the requirements and the interactions between the buffer and other elements of the system is clearly necessary, the tentative capacity required in each buffer storage unit is in the 10^5-10^6 bit range, with a cycle time of 100 ns. The relation between these numbers and those of the other modules in the system will be developed in the discussion below.

5.3 Processor

To specify the power of a processor in terms of millions of instructions per second (mips) inherently requires the existence of a standard for definition of the "instruction" itself. Neither this standard nor the instruction set for this processor have been developed. However, despite the ambiguity of the measure, we will assume that processors should have a .5 to 5 mips capacity to meet the space station requirements. The processor configuration proposed by Intermetrics includes an alterable microprogram, and is organized around a stack concept resembling the Burroughs computer family, particularly the 6500/7500 processors (cf. Chapter 3, and Appendix A).

5.3.1 Microprogram Characteristics

The 360/85 microprogram consists of about 2500 108-bit words, or 2.7×10^5 bits[10]. The 360/25 has about half that number, while the Standard IC-9000 offers up to 2×10^6 bits[14]. While the specification of the space station computer's microprogram size must be postponed until a later design phase, it is reasonable to assume that it will fall in the $10^5 - 10^6$ bit range, with access time on the order of 50 ns for the high-performance version.

Part of this microprogram should be implemented in fixed memory, the exact fraction to be determined later. However, it is necessary at a minimum to include in the fixed part those microinstructions required to load the variable part, plus those to perform a "dead start". Further, certain diagnostic functions should be included, such as ones required to isolate a problem which prevents successful loading of other diagnostics into the variable part.

At least a part of the microprogram should be implemented in read-write storage, or be switchable extensions in read-only memory, or both. In addition to the advantages of conventional microprogramming mentioned in section 3.8, use of the alterable or switchable portion allows the user to capitalize on the dynamic modification of the apparent characteristics of the processor. This would permit processes, for example, to be executed on processors having high-efficiency floating point operations, character-string manipulations, list-processing instructions, etc., at will. A further generalization would allow the instruction set of a processor to be closely tuned to the characteristics of each of a number of higher-order languages.

5.3.2 Stack

Because the stack plays such an important role in the re-

duction of internal-bus traffic, it presumably would be implemented in the fixed part of the microprogram. Further, it provides some of the primary guidelines for processor design. The fundamental utility of the stack lies in its automatic ability to dynamically allocate and deallocate storage locations for temporary values from a pool provided for the purpose. This results in extensive time-sharing of these pool locations, but without requiring explicit assignment action by programmers, an error-prone activity. Since the result of this organization is to produce intense load/store activity near the top of the stack, the stack is unusually powerful in the proposed organization, since the provision of a number of registers in the processor, with an extension in a dedicated portion of the buffer, keeps a high percentage of fetches and stores from using the internal bus. A processor's use of the stack is an implicit declaration that the quantities involved are local to the process; there is thus no need to keep an M2-memory copy of them for potential use by other processes.

The number of locations that this concept requires is an undetermined design parameter at this time. The B6500 design includes only two stack slots and a pointer in processor storage; however, a larger total is obviously necessary, to achieve the bus-traffic reduction sought. To allow maximum stack size to be independent of the number of locations provided in the M1 memory, stack extension into M2 should be possible. Thus, M1 would initiate stores to M2 when the area became filled or when stack-switching occurred.

5.4 Segmentation, Paging, and Level-2 Memory

5.4.1 Segmentation and Paging Design

Section 3.5 described the storage fragmentation problem, which has required increasing system-designer attention since the one-job-at-a-time days of computer operation. The three means currently most often used to either avoid or deal with this condition are:

1) Use of relocation registers in the processor. Since only these few relocation values (one or two per process) represent the translation between logical (virtual machine) and physical storage addresses, it is relatively easy to interrupt running processes long enough to "repack" core to eliminate the fragments when necessary. Modification of only the affected relocation values is sufficient to complete the procedure.

2) Use of paging. This technique assures that the fragments of free space and occupied space are of uniform size; since address translation makes scattered pages look

logically contiguous, the only unusable space which can occur is that within partially empty pages.

3) Segmentation. Although segmentation can be considered as a logically separate concept from paging, the success of the Burroughs 5500 and 6500 systems indicates that it is not necessary to do so. Clearly, if the typical segment size is a small fraction of the storage area from which its space must be allocated, the amount of space lost to fragmentation will be acceptably small. Then segments (except for very large ones) may be alternatively considered to be variable-length pages, and treated as such by the hardware and software alike.

It is the third of these approaches which has been selected for the present design. Segmentation of program into relatively small units will be automatically performed by the higher-order language translators. There is overhead caused by the indirect addressing required during execution (analogous to the segment-table access in a segmented-and-paged system, but without the "page" table access, except in a few cases); however, the high speed and automatic loading characteristics of the buffer memory promise to reduce the effective cost of this to quite an acceptable level, since the indirect address words are likely to be found in the high-speed buffer. As mentioned, this approach does require treatment of exceptional cases: "paging" of very large segments, and storage reassignment for unpaged segments which outgrow their currently allocated storage. However, because the number of segments large enough to require paging and the number of segments of dynamically varying size are both small, the system as a whole benefits from the more streamlined design suited to the vast majority of cases.

5.4.2 Level-2 Memory

Section 3.5 presents in Figure 3.13, a graphic illustration of the ill-effects of high traverse time for a required block of words from a lower-level store. For a typical T for drum-core transfer of 1.5×10^4, and a missing page probability of 0.003 (1 page fault per 300 instructions; cf. 3.5.4.2), the paging efficiency, computed as in 3.5.4.1, is a skimpy 2.2%.

To achieve a considerable improvement of efficiency, within an assumed groundrule prohibiting use of memory devices with moving parts, it is proposed to provide a maximum level-2 memory (M2) of 1 μsec cycle time and $10^7 - 10^8$ bits capacity. This memory will be used in the way both drum and main memory are used in contemporary systems; programs will reside there when required by one or more active processes, and will be executed from there by running processes. Not only does this strategy enormously improve the "paging" efficiency by making fetches from M3 memory occur only once per process per segment, but

it eliminates processor overhead experienced in the storage allocation program associated with swapping. This overhead averages 3 - 6 milliseconds per page fault in the current Multics implementation[9,13].

With respect to modularity and interleaving, one reliability and two operational requirements dominate the design decision. Modularity is necessary to allow for systematic expandability. Interleaving is desired within the module to allow reduction of the traverse time from M2 to M1 below the one memory-cycle per "word" that otherwise would occur. The level of interleaving proposed is that which allows a block fetch by M1 to be answered by one module of M2 at the maximum rate M1 can accept. This in turn depends on the number of bits accessed per M2 sub-module fetch, a parameter not yet determined.

However, unless the contents of memory are adequately protected against loss, n-fold interleaving magnifies the region of a unit failure by the same n-factor. More is said of the contents-loss problem in section 5.7.

5.5 Level-3 Memory

The level-3 or M3 memory in the proposed design consists of two independent sections: read-only, and read-write. The characteristics of the mass read-only memory required on the space station are expected to depend primarily on the reference requirements of the experiment packages. These requirements are even more elusive to grasp at the current time than are estimates of data processing needs, and therefore no attempt will be made to size this memory. Two of its other design features are more readily specified: first, the speed and addressing properties of the ROM should be about the same as the read-write part of M3, for system compatibility. Second, the ROM should permit introduction of new data loads in at least a fraction of its address space. The characteristics of this memory will be more extensively discussed in the report of the Mass Memory Study which is part of the current contract.

The read-write part of M3 is analogous to the disk and data-cell storage found on commercial computers. This memory holds those programs and data files which are available for use by system users. A block-oriented configuration would be suitable for this level, if any advantage could be gained by this approach. Contents will be located by means of a directory hierarchy similar to that used in current systems, such as OS/360 or Multics. The maximum size of the M3 level of read-write memory is estimated to be on the order of 10^9 - 10^{10} bits; its speed should be such that the traverse time ratio for a block of words into M2 falls in the 10 - 1000 range, but this parameter is felt to be fairly unimportant.

5.6 Data Transmission

5.6.1 Processor to M1

The architecture described in section 5.1 relies heavily on the satisfactory communication of data between several hierarchical levels of memory. A system processing potential of up to 10^7 instructions per second is proposed. This implies that the processor must be able to access the M1 memory for data within about 100 nanoseconds. This data path is a dedicated one; it need not be shared by other processors or memories. It can therefore be designed without consideration for the flexibility and expandability requirements of the system as a whole. The physical distances over which the processor and M1 communicate will be of the order of inches, rather than feet. This can be achieved by carefully designed conventional wiring, and will not necessitate exotic transmission line techniques. Depending on the chosen "width" of the path, i.e., the number of separate parallel lines constituting the bus, data and control pulses of 10 to 100 ns duration will be involved, at frequencies between 10 and 100 MHz. The major problems are:

a) Maintaining the shape of a pulse as it traverse the interface. This is a matter of matching the distributed capacitance and inductance of each line, and terminating each in its characteristic impedance to avoid the reflections which would otherwise degenerate the next pulse in sequence.

b) Maintaining equal transmission delays over the individual lines of a parallel interface. This may involve the "looping" of connecting lines between terminals that are closer together than others. Synchronization is important in achieving high repetition rates in parallel data transfers.

c) Minimizing the effects of undesired coupling between adjacent lines. A common measure is to surround each conductor with a low impedance ground shield, or to sandwich it between parallel grounded conductors. Unfortunately, this adds to the capacitance in the system and therefore increases the power required to drive the lines. For a specific design, a compromise between physical topology, speed, and power dissipation must be reached.

The design of the IBM 360/85 cache memory[2] illustrates one current approach to some of these problems. This memory is about one cubic foot in volume, and has interconnection distances of a few inches. It is organized into cards which carry the storage, drive, and sense circuits. Each card is designed to provide equal transmission delays for parallel data paths: delay times to all storage elements are within 3 ns of each other. Space on each card is devoted to termination resistances for the X and Y drive lines. Although the individual circuits have

characteristic delays of less than 10 ns, the delay per board averages 25 ns due to the contribution from the equalized wiring. The overall memory of 0.25×10^6 bits has 40 ns access and 60 ns cycle times, which are comparable to those required for the maximum computer configuration proposed in this report. Developments in packaging density and circuit power requirements can be expected to assist in the achievement of high performance transmission paths between the memory and the processor.

5.6.2 The Internal Bus

The transmission of data between processor/memory pairs and other elements of the system such as mass store, I/O units, etc., may involve physical distances of up to a few feet. A data bus is proposed to link all these elements. Because multiple data will pass along a common physical transmission medium, a multiplexing approach is necessary. The data bus itself becomes a significant element in the computer and due consideration must be given to its design.

The peak data rate requirement comes from traffic between M1 and M2 memories. To support the proposed top processing speed, information must be delivered from M2 at the rate of a single word transfer in 100 ns. The bit rate capacity that this requirement imposes on the bus is a complex question. Some of the factors are enumerated here:

a) Message Structure. Since many types of data will be transmitted on the bus, identification and control are necessary. This implies either separate control lines, complicating the bus structure, or addition of control bits to the basic data word, thus increasing the required bit frequency.

b) Bus-access Control. In time-division multiplexing (which is the probable approach), only one message may appear on the bus at a time. A technique must be devised to grant bus-access to units seeking to transmit data. Conceptually, perhaps the simplest way is to sample each unit at a rate high enough to allow satisfactory dynamic operation, with additional contingency for expansion. The direct effect of this technique is to force the required bit rate up. Other techniques, such as the request-and-grant approach, ease the bit rate requirement, but impose further hardware complexity.

c) Error Checking. Techniques for detecting and correcting errors, e.g., Hamming or other transmission codes, add more bits to the message. It may, however, be possible to combine memory and bus error checking as a part of the bus system, since they have certain similar characteristics.

d) Reliability. The likelihood of failure is generally proportional to system complexity, other things being equal. A bus consisting of a single wire is inherently more reliable than one comprising a hundred. For a given level of information traffic, however, fewer lines demand a higher bit rate capability.

e) Bus Interface Circuitry. The elements of the computer system generally process data in a parallel, word- or byte-organized fashion. (High speed linear-select memories handle especially wide data structures, sometimes up to several hundred bits, to minimize internal drive problems.) Serial/parallel conversion is necessary for elements to interface with a bus system that is narrower than the basic data. For very narrow buses, very fast circuitry is necessary to perform the conversion, with the attendant problems of layout, power dissipation, etc.

The proposed computer design imposes a peak memory-to-memory transfer load on the bus of about 40 information bits per 100 ns interval. To this must be added a block transfer overhead of about 20-30 bits of address information, up to 10 bits of message check-coding, and, say, 20 bits of bus identification and control overhead. For eight-word blocks, this may total 400 bits in an 800 ms interval. If a minimum pulse half-period of 10 ns is postulated, this data rate can be accommodated on a 10-path parallel data bus. Since about 2.2 Hz bandwidth is required per pulse per second, the minimum bandwidth necessary to achieve transmission of 10 ns pulses is approximated by $2.2/(10 \times 10^{-9})$ or 220 MHz. This is within the capability of miniature coaxial cable, which can handle up to 500 MHz. More lines than 10 would allow more straightforward wiring techniques to be considered (dependent on length), but would increase the complexity and interconnection problems, and decrease reliability.

Providing connections to transmission lines involves problems of impedance matching, attenuation, noise suppression, and level conversion. A technique employed in the IBM 360/85, the directional coupler (well known in microwave technology) has been described.[3,4]

The directional coupler enables simple junctions to be made to transmission lines without the usual line-to-line impedance mismatch and its attendant restriction on the length of the junction. It requires no DC connection, eliminating voltage level and grounding problems. It inherently suppresses driver noise and mismatch reflection by virtue of its directionality. Its chief disadvantage is that its operation relies on the steepness of the transmitted pulse, which must induce corresponding voltage changes in a capacitatively coupled conductor. Both rise and fall times must be in the subnanosecond region. This imposes the usual layout and power dissipation problems on the drive

circuitry. However, integrated circuit techniques being developed today are probing the fractional nanosecond region[8], and by the time period projected for the operation of the proposed computer, compact techniques for driving a line with sharp pulses of sufficient energy are expected to have been developed.

5.7 Error Detection

The reliability concept for the space station DMCS includes not only the conventional notion of low probability of component failure, but additionally requires uninterrupted error-free performance even when components ordinariy considered to be critical have failed. It is inconceivable that techniques will be available by the mid 70's for building a computer which can operate for five to ten years without repair, even if massive redundancy is used. Consequently, the reliability of the computer system must be such that the number of failures which occur in the time interval between the occurrence of the first failure and the repair of the system does not exceed the failure tolerance capability built into the design. Because of space and weight limitations in the space station, it may be impractical to provide on-board spare modules for each system. Thus, the reliability and failure tolerance goals may need to be keyed to shuttle flight schedules rather than derived from the expected time to diagnose and repair a failed module.

The soundest approach to reliability in the technology of the foreseeable future is to use simple, conservative circuit design and the most reliable parts and fabrication techniques available. Only when this approach fails to yield the required reliability should the designer resort to redundancy; the addition of redundant components introduces additional failure possibilities which tend to offset their effectiveness. In fact, attempts to achieve unrealistic reliability goals may cause the product to contain such complexity that the failures against which the design is meant to be protected occur in the protection equipment itself with such high probability that only the cost, and not the reliability, is found to have increased. Therefore, any imposed requirements for multiple-failure tolerance should be reviewed in particular because of the impact it has on the design.

Methods for detection of failures when they do occur may vary considerably between the types of units because of the disparity in their functional characteristics. The buses, for example, lend themselves to checking by use of transmission codes, of which a great deal is known. Many of these codes permit errors to be corrected as well as detected, so that transient failures may be rather easily masked. Processors, however, modify data they handle in so many ways that checking, though possible, is more difficult. Recovery from transients is straightforward if inputs to operations are retained, since the

operation which failed may then be repeated. Memories have some of the characteristics of buses, and transmission codes may be used for detection and correction of certain errors. However, other failure modes cause contents to be lost; when data is destroyed, only reconstruction by some process can permit full recovery.

The proposed multiprocessor configurations inherently are tolerant of processor failures if four conditions are satisfied:

a) All processor-errors are detected, and the system advised.

b) A failed processor can be logically removed from operation, and does not contaminate the system.

c) Sufficient processing capacity remains after the failed units have become dormant.

d) The hardware/software combination can reconstitute the process which was running on the processor at the instant it committed the error.

Section 3.4 demonstrates the relative effect on availability produced by the use of redundant elements.

Memory failures differ conceptually from processor failures, since they represent a potential loss of information, rather than a loss of capability. Memories tend to fail in four ways:

1) One or more words read have a bit in error;

2) when a word is addressed, no response occurs;

3) when a word is addressed, the superimposed contents of several words are delivered;

4) when a word is addressed, the contents of the wrong location are delivered.

The first of these may be handled by provision of a transmission-type code such as parity or a cyclic block code. Such codes are relatively easy to implement, and can provide both detection and correction capability. If the all-zero bit combination is an error code under the chosen method, the second type of failure above is detected. The third is more difficult to discern because of the possibility that superimposed words may pass the checks. For simple parity, use of a word with an even total number of bits and odd parity is the best solution. However, if the memory fetches two words, the chance of this error being detected is only 50%, which is clearly unsatisfactory. More elaborate checking codes, such as Hamming codes, detect superimposed words with higher probability. The fourth type of failure cannot be detected by any of these methods,

and protection against it involves further complexity, such as storage of address bits along with data, for example.

Thus, it presently appears desirable to operate multiple copies of memory simultaneously. All write commands would be accepted by all functional copies, so that each contains completely current contents. Each will respond to read requests, and all outputs will be compared in an evaluation circuit. Provision of checking codes permits the detection of the erroneous word in most cases. However, in the few instances where this fails, the software is required to attempt recovery. The overall result is the elimination of undetected errors and the resulting propagation of bad data throughout the system.

If the redundant-memory-copy approach is adopted, some cost saving may be realized by operating some modules of M2 in simplex mode. This is possible because all program segments are pure procedures, and can thus be recopied from M3 if the M2 area they occupy should fail. With this philosophy, it is only necessary to detect errors in M2; correction within that domain of M2 is unnecessary.

Intermetrics recognizes that there is a potential hardware/software tradeoff in the attainment of reliability. The approach outlined above reflects our strong belief that hardware failures in the computer system must be rendered invisible to the applications software whenever possible. As a result of comprehensive experience in the development of the Apollo on-board G&N software, we have found that trying to achieve failure protection primarily by software techniques is incredibly expensive. Further, in addition to the steeply increased software cost, there is invariably an associated software unreliability which remains even after completion of extensive and ambitious testing, which prevents attainment of the overall system reliability sought. The redundant-copy approach outlined above reflects this experience. Nevertheless, we are aware of the cost, power, weight, and volume penalties which result from the provision of redundant copies of units; we submit that this area must receive additional study.

5.8 Operating System Philosophy

The computing system for the space station is required to accomplish a spectrum of activities which includes real-time control, general-purpose data processing, and interactive computing directed from remote terminals. It must also allow users to share programs and data. These requirements span virtually the entire range of computing problem types, which usually are performed in computers dedicated to a single one of these functions. Consequently, the operating system must be a very broad and general one, and yet must not cost the system an unreasonable amount of overhead.

The general characteristics and philosophy of the operating system are summarized below:

1) The operating system will contain the conventional kernel of programs required to run the entire computing system itself. This includes scheduling and dispatching of tasks or processes, a dynamic relocation mechanism for management of information transfer between the M2 and M3 memories, and a comprehensive file system. The file system must be sufficiently general to allow data sharing and data interlocking among users, although it is probably not appropriate to require the file system itself to perform the interlocking.

2) It should be the responsibility of those who prepare the operating system to also prepare a comprehensive set of system utilities for use by application programs. These include display interface routines, language processors, and so on. It should be within the capabilities of the protection mechanisms of the operating system to prevent other than specified system routines from being used, for example, to command input/output devices or displays, to prevent proliferation of interface programs, and also to prevent misuse of the devices.

3) The system must provide the maximum achievable autonomy for users. This is desirable both to allow decentralized application-programming efforts, with local management, and also to prevent a continuing requirement for augmentation of the operating system to meet new and changing requirements. This autonomy does not preclude or even discourage exercise of good management practices over the overall programming job. Specifically, it makes it a fairly straightforward job to apply memory space and execution time budgets to the decentralized autonomous functions.

4) In conjunction with the philosophy of decentralized programming, the users themselves must be required to provide the software to handle their own equipment on board the station. That is, no specific I/O routines for special-purpose user equipment will be included in the central supervisor. However, it is desirable to make I/O instructions "privileged", so that the supervisor can validate them in order to control accidental access to unauthorized devices or memory. Access to programs and data should be granted by this operating system on the basis of the identity of the individual user. That is, each program and data file should have access rights or access control information specifying who may, or who may not, have read, write, or execution access. The identities used by the access controller should be functionally oriented; that is, access privileges should be based on the function being performed, rather than the identity of the individual, assuming that the system has already verified the particular individual's authority to perform the function.

5) Any function required to effect recovery from computer system hardware failures or errors, but which is not implemented in the hardware itself, must be performed by the operating system. In no case should any computer failure require application-software recovery action. On the other hand, failures of application-hardware are outside the operating system domain.

5.9 Word-length, Protection, and Flag Bits

5.9.1 Word-length

At the very heart of any discussion of word-length lies the question of what the function of a word is. Indeed, not all computers are word-organized: some computers are character-oriented with variable length instructions; punctuation bits associated with each character (word marks and item marks) denote the end of data. (IBM 1400 series is an example.) Other computers are byte-oriented, the term referring to the absence of punctuation bits. But the word-oriented computer still finds much favor. While the byte-oriented computer regards data as strings of characters that can be manipulated as strings or individual bytes, the word-oriented computer organizes the data into fixed-length groups. Groups of words may make up an array. Floating-point data fits neatly into words, as do fixed-point and integer quantities. In fact, fast arithmetic operations are more directly implemented with word-parallel hardware; variable-length operands typically require digit-serial operations. Alphanumeric and string data are best suited to byte or character representation. Of course, string data can be formatted on a word machine, and arithmetic data can be represented on a character machine. The question is one of relative gains and losses.

Recognizing the hazards of premature design decisions, but also seeing the need for a design strawman, it is proposed that this computer should be word-oriented.

Within a word there must be a smaller unit to represent characters of information. Although it can be argued that eight bits is wasteful, the eight-bit byte has received an acceptance that is approaching universality in recent computer configurations. Special cases and exception-handling seem particularly inappropriate in communication equipment and transmission paths; since these seem inevitable if shorter bytes are used, the 8 bit byte is chosen.

In selecting the word-length, it seems mandatory that it be an exact multiple of the byte length. Reasonable arguments can be generated that support a word length of 24, 32, 40, 48,

or 64 bits. Actually, the field can be reduced to two fundamental choices, either 32 or 48 bits. From the point of view of scientific computation, the scales are tipped in favor of 48 bits. Experience has shown that most floating-point problems can comfortably live within 48 bits with double-precision required only for rare exceptional cases. However, 32 bits is too small for a large class of routine calculations required in aerospace applications. Thus, a long version of 64 bits would be often required. Occasionally, an exceptional case would still arise which requires even longer precision. Hence, at least three different floating-point number representations are needed.

Nevertheless, 32 bits has been selected for the word length. Although it is convenient to have a word that contains a number of bytes which is a power of two, this is not an over-riding consideration. For the last five years, any computer introduced that has a word length which is not multiple or submultiple of 32 bits (8, 16, 32, or 64 bits) appears eccentric. Thus, again the decision is motivated by considerations of compatibility and general standardization. This compatibility permits ease of simulation on existing ground-based computers, and allows a direct comparison of arithmetic results for both floating-point and fixed-point calculations, assuming that the number system, floating-point formats, and algorithms are similar.

5.9.2 Memory Protection

Memory protection can be achieved in a very rigorous, yet flexible manner, through a combination of hardware mechanisms, rooted in the basic structure of the computer, and software capabilities exercisable only by the operating system. The fundamental concept adopted by Intermetrics to protect one program from undesired access by another is one of prevention rather than detection. One process cannot adversely affect another if there is no way for them to address each other. The variables and information of one are simply placed outside the scope of the others.

Some of the characteristics of the computer that produce this environment include the following:

a) Absolute addressing may never be used by application programs. All addressing is relative: relative to the base of the procedure or subroutine segment, relative to the base of the array or buffer areas, relative to the base of the stack or the stack pointer. Thus, all addressing is oriented to key items that have been assigned to the process by the supervisor. As a bonus, this addressing is automatically relocatable.

b) Upper limits are imposed, as are lower limits or base values. When the executive assigns an area for data or program or a

process, it also places an upper bound on the segment, thereby constraining the process to stay within its limits. Hardware is provided that automatically detects attempts to go beyond the allocated area, by incorrect indexing for example. Thus, operating windows are established, and programs are confined to remain within the windows.

c) The linkage words, indirect addresses, data and program points, and supervisor calls cannot be modified in the problem state. The scope of knowledge of any operating process is thus established by the executive, and fully controlled by it. No individual program can escape its own region.

d) The links form a small and carefully controlled network of interconnections between various programs and data. The executive can remove or modify these links to adjust the capabilities of individual tasks, or to restructure the memory allocation. This gating of the interfacing between program elements through limited connecting paths provides a powerful means of regulation to prevent inadvertent accesses.

e) Finally, individual words are identified and protected by a set of bits, called flag bits, which are described in detail in the next section. It is these bits that protect the link words from being overwritten by problem state programs.

5.9.3 Flag Bits

Associated with every word of memory are several extra bits that identify the type of word, and offer a means to prevent it from being accidently modified or incorrectly used. Current design thinking suggests the following developmental strawman:

a) Three flag bits will accompany each 64-bit double-word. The three bits will be examined by the hardware whenever the whole double-word or either single word is fetched or stored.

b) The flag bits apply to both words, so that both must be write-protected when one must be.

c) Functions performed include write-protection, trace mode, and identification of variables, constants, program, interlock words, and shared data base pointers.

d) Proposed bit-pattern categories:

   1) Writable variable

   2) Trace-trap this variable

   3) Interlock word, used for shared-data handling.

4) Instructions or constants

5) Trace-trap this instruction or constant

6) Special double-word: link word, pointer, etc. Flag sub-bits will further identify.

The last three are write-protected.

The class of special double-words are useful in many ways. Among other things, they offer a method to provide an indirect address to data when data itself was expected, to produce a jump to a closed subroutine when data or an address of data was expected, or to cause an interrupt to the executive under other conditions; e.g., in order to allocate a data area in memory. These special double-words are generally only created and updated by the executive. However, certain types of them are freely produced and altered by specific problem state instructions, such as subroutine call and return.

Special double-words are used for:

a) Indirect addresses: expected pointers to data or program.

b) Unexpected pointers

1) Data pointers when data expected (indirect address)

2) Program pointers when data or data pointers expected.

c) Interrupt calls

1) Missing data, program, or data area

2) Supervisor-calls, or entrance to executive routines

d) Special words

1) Stack markers

2) Link words

3) Pointer-pointers: addresses in stack of executive-generated pointers; e.g., subroutine return markers

The trace-trap condition causes a trap to trace routines to indicate whenever an instruction or operand is executed or referenced. This is an extremely useful debugging tool.

The interlock word indicates a type of word that is treated by the hardware in a unique fashion. It provides a mechanism to accomplish interlocking that is done by the Test and Set instruc-

tion on many computers. It is also designed to facilitate the sharing of data. It operates as follows:

a) The interlock double-word serves as a pointer to a resource whose usage needs to be regulated or interlocked. Users are gated through this control or check-point.

b) A lock instruction examines the interlock double-word, and if the lock is free, it is locked by the placement of two pieces of identification in the lock word. One is hardware-determined; the other can be varied by the program for internal communication. If the lock is busy, the program either loops or idles until the lock is free.

c) An unlock instruction reverses the process; it frees the lock and returns the word to the available state.

d) When the lock is busy, usage of the pointer to access the interlocked resource is prohibited by hardware unless the user's identification key matches to an acceptable degree the lock value.

e) For simultaneous reading of shared data bases a special halflock instruction is provided. This locks the resource from store instructions (write-protect) but permits other programs to read the data. Two or more users may make successive halflock requests which are accepted. The suggested procedure is to combine part of their lock ID's by a simple process, say an exclusive OR operation. Then an unlock or halfunlock could do another exclusive OR of the ID of that process.

f) An instruction desiring to store into a shared area must be preceded by a full lock instruction, which would stall until all other users are through reading the shared data as indicated by a free lock. Then it would lock it for its exclusive use, and do the updating of the shared data. When through, it would unlock it for others to use again.

Other uses of the interlocking mechanism will be uncovered as the software system is developed. The synchronization and control of concurrent processes is an immediate candidate.

5.10 I/O and Interrupt Structure

Communication with its external environment is an essential facet of the operation of the data management computer system. Nevertheless, consideration of I/O requirements and implementation formed a very minor part of this study. Only broad design conclusions were reached; they are presented in the following sections.

5.10.1 I/O Bus

It is postulated that the space station communications with the DMCS take place over a limited number of lines, perhaps as few as one. These system I/O buses would be routed throughout the spacecraft; units would communicate with the bus through a standard interface element, which would provide the necessary isolation in case of failure, and be capable of sending and receiving messages in standard but variable-length formats. A bit-rate capacity limit of 1-10 megabits per second is visualized for the I/O buses to allow their construction to be elementary and their reliability to be exceedingly high. Should a unit be capable of overloading the I/O bus, either a special bus should be provided, or preferably, data compression techniques be applied at the unit to lower the actual bit rate. At the computer end of the bus, communication takes place via one or more I/O controller units, described in the following section.

5.10.2 I/O Controllers and Interrupts

Because of the variety of I/O device types which can be anticipated in an orbiting scientific facility, the I/O controller must be a generalized and multiplexed unit with a simple communication interface. A typical I/O operation is one in which a processor issues a request to an I/O controller for a specified block of data from a device. This request, transmitted to the IOC on the system internal bus, specifies the device, an action code for the device, a word count for the data block, and the address in M2 where the block is to be stored. An additional bit is used to specify whether a completion-interrupt is to be generated by the IOC.

Upon receipt of this request from the processor, the IOC in turn issues a request to the device over the appropriate I/O bus. The device, at its own response rate, replies with an identifier and the block of data. The IOC directs this data to M2, and when the transfer is complete, signals the requesting processor via an interrupt message, if one was requested. Multiplexing in the IOC permits many such operations to be in various states of completion at the same time. Because the IOC retains the identity of the requesting processor, any error signal which occurs may be transmitted in an interrupt message to that processor for action.

To allow I/O devices to volunteer data without being specifically requested to do so, it must be possible for a device to initiate a request-interrupt, which would contain a code indicating the action desired of the processor. Because this interrupt is not a result of a processor-initiated operation, no processor identity will be held in the IOC to be used as the target for the interrupt. The IOC can either select a processor at random, or address the processor which last initiated a command to the device.

References for Chapter 5

1. Ashley, D.W., "A Methodology for Large Systems Performance Prediction", IBM Technical Report TR 00.1773, Sept. 10, 1968.

2. Ayling, J.D., and Moore, R.D., "A High-Performance Monolithic Store", Digest of Technical Papers, 1969 IEEE International Solid State Circuits Conference, p. 36.

3. Bolt, M.H., and Nick, H.H., "A New Tool in Computer Bussing: The Multiplex Directional Coupler", Computer Design, May 1969, p. 40.

4. Private communication with M.H. Bolt, IBM, Durham, North Carolina. March 17, 1970.

5. Conti, C.J., Gibson, D.H., and Pitkowsky, S.H., "Structural Aspects of the System/360 Model 85: General Organization", IBM Sys. J., Vol. 7, No. 1, 1968.

6. Denning, P.J., "The Working Set Model for Program Behavior", CACM 11,5. May 1969. pp. 323-333.

7. Gibson, D.H., Shevel, W.L., "'Cache' Turns Up a Treasure", Electronics, October 13, 1969. p. 105-107.

8. Gold, H.S., and Pedersen, R.A., "An Integrated Logic Gate with Subnanosecond Propagation Delay as a System Element", Digest of Technical Papers, 1969, IEEE International Solid State Circuits Conference, p. 70.

9. Informal presentation by J.M. Grochow at the Multics Users Forum, March 4, 1970.

10. IBM System/360 Model 85 Functional Characteristics, Form A22-6916-1. June 1968.

11. IBM System/360 Model 195 Functional Characteristics, Form A22-6943-0. August 1969.

12. Liptay, J.S., "Structural Aspects of the System/360 Model 85: The Cache", IBM System Journal, Vol. 7, No. 1, 1968.

13. Organick, E.J., "A Guide to Multics for Subsystem Writers", Chapter VII, p. 7-28, MIT Project Mac Memo M0115, August 1969.

14. Rosin, R.F., "Contemporary Concepts of Microprogramming and Emulation", Computing Surveys, Vol. 1, No. 4, Dec. 1969, p. 196-212.

Chapter 6

Summary and Recommendations

6.0 Introduction

This chapter is included to summarize and comment on the information collected and analyzed during the course of the study, to discuss technology trends and their impact on the computer, and to present areas in which it is recommended that more study or design occur.

We believe that the overall objectives of this study have been achieved. Existing knowledge concerning actual and proposed multiprocessor systems has been collected and included in the survey. General multiprocessor theory and a baseline definition of various system configurations has been presented and analyzed in Chapter 3, along with a discussion of the design considerations and a review of the existing space station DMS requirements. An architectural design of a multiprocessor computer system was presented as an extension of multiprocessor technology to the space station DMS application.

6.1 Technology Trends

The computer design presented in Chapter 5 can be satisfactorily implemented for a space base application only if certain technological developments take place. A computer of this design could certainly be built using current techniques, but its probable volume, weight, power consumption, and environmental requirements would be more comparable to a large data processing facility, which it resembles in performance, than to the small aerospace computers currently in operation. Without considering the M3 memory, the physical characteristics of the system might be: volume, 500-1000 cubic feet; weight, several thousand pounds; and power, 50-100 kilowatts. Appendix B makes a tenuous estimate of the physical characteristics of the computer for a 1975 cut-off, based on a continuous development of today's technology. Predicting what these developments will be is a hazardous task. However, it is possible to discuss trends that are visible today and to identify areas of the computer design that should receive the major emphasis for improvement.

6.1.1 Memory

The greatest reward for the effort of development will come in the area of memory, which is generally the bulkiest, costliest, and slowest element of a computer. The ferrite core has dominated the main high-speed memory (capacity up to 10^7

bits) for so long since its first application in the mid-1950s that "core" has become synonymous with "memory". Its "imminent demise" at the hands of up and coming competitive techniques (e.g., plated wire, film) has been heralded for ten years; core technology has always responded by increasing its dominance in this application. Early cores were large, 100 mil diameter, and slow, 10 μs switch time. Successive improvements have occurred in both the geometry of the core, which is in commercial production in a 12.5 mil O.D., 7 mil I.D. size as the element of a 375 ns cycle time memory[9], and in the organization of the memory core arrays[2]. These improvements have enabled cores to at least keep up in performance while always maintaining a steady lead in terms of cost.

Despite its past success, the ferrite core is bound to yield its position sooner or later. There are several reasons for this view today: Firstly, the bit density of core arrays will never match that obtainable by other techniques, such as semiconductors. The high speed memory of Ref. 9, for example, has an overall density of 4,000 bits/cu. in., which is 10 to 100 times less dense than current MOS techniques. Since memory capacity requirements are expected to go up with time, this factor will become increasingly important. Plated wire and thin film memories, although potentially an order of magnitude faster than ferrite core, also suffer from a poor bit density: 1000 to 10,000 bits per cubic inch. In spite of a current strong emphasis in developing plated wire,[4] it is our opinion that this technique does not have the ultimate promise of semiconductors.

Secondly, the performance curve is beginning to flatten out at about 100-200 ns cycle time. Techniques such as even smaller cores, partial switching, two cores per bit, or expensive 2-D configurations must be employed to get into this speed region; these are all factors that diminish the ferrite core's advantages of simplicity, manufacturing ease, and cheapness.

Thirdly, the competition is in a very healthy developmental phase. Semiconductor memory arrays are currently the subject of intense commercial activity; this is always the portent of general acceptance of a new technology. LSI arrays of bipolar transistor flip-flop memory cells provide the capability of very high speed (10 - 100 ns) local random access memories of limited capacity (up to 10^5 bits). Currently, densities of about 100 bits per 100 x 100 mil chip with dissipations of about 10 mw per bit are being achieved. MOS transistor arrays provide denser packing at a lower speed (1000 bits per chip at 1 μs) and lower power dissipation (as low as 10 nw/bit for CMOS logic), for implementation of the larger main memories. Other technologies, such as the MNOS transistor[8] which has the property of non-volatility and which requires only one active device per bit, and the magnetic domain "bubble" in orthoferrites,[1] promise very high densities, and are the subjects of current

experimental investigations. The rate of development of semiconductor techniques, which promise orders of magnitude improvements in speed and density, is accelerating, whereas current memory technologies are reaching a developmental plateau. It seems inevitable that the memory system of the next generation of aerospace computers will be realized to a great extent with semiconductor techniques.

6.1.2 Logic

Improvement in the logic areas of the computer system, the processors, I/O controllers, buses, etc., have until recently been realized by increased speed in the logic and control circuitry. However, the logic available today already operates in the sub-nanosecond region[5]. The problem now is the timely communication of data at these high speeds over physical paths[3]: a pulse is delayed by about 2 ns for every foot of interconnection. The reduction of physical dimension and the minimization of interconnection paths becomes the route to increased circuit performance. The possibilities offered by large scale semiconductor integration in pursuit of these ends are overwhelmingly attractive and consequently LSI has received a great deal of attention and publicity, some of it premature.[6] In conjunction with increased circuit speeds, performance improvements will be achieved by tailoring the organization of the computer to this end. The concept of the buffer memory proposed in Chapter 5 is an example. It enables large programs stored in the relatively slow M2 memory to be executed at a speed approaching the capability of the much smaller high speed memory M1. The use of parallel rather than serial processing is a path to increased performance which has always exacted penalties of increased complexity and cost. However, it is just these factors that LSI seems particularly suited to combat. With the increased employment of LSI techniques, we would expect to see highly parallel logical organization in the design of future aerospace computers.

6.1.3 Summary and Recommendations

Technological improvements are most needed for the implementation of the proposed computer memory system. It is anticipated that LSI semiconductor techniques will be the main line of attack. If improvements in LSI yield (cost), speed, power dissipation, and density continue to accelerate, this technique will be used in the majority of the logic and memory circuitry in computers beyond 1975.

It is, however, a long road from the experimental demonstration of a promising technique to its application in a real operational environment. Although there is much activity, for example, in semiconductor chip design for high yield, high density and low dissipation, the problems of integrating these

chip elements into an operational unit such as a 10^8 bit memory are not being as widely pursued. We would recommend that implementational problems associated with interconnecting a large number of densely-packed elements be evaluated in a pilot study. The construction of an operational memory should be undertaken with a view to identifying the construction technique that achieved the best compromise between performance, cost, and size. Recommendations as to the best technology and memory size for the DMS computer will be made at the conclusion of the mass memory study that is a part of this contract.

A vital part of the proposed computer configuration is the transmission of data between the various elements along a common data bus. The many data types, resulting in complex bus message structures, and the high speeds will make severe demands on the implementational technology. The interfacing of the bus with the processing elements will involve high speed, serial-parallel data conversion, error detection and correction and signal conditioning and re-formatting. We would recommend that the bus, and its interfaces be the subject of separate study and experimental implementation.

These studies should precede detailed DMS computer design work by 1-2 years in order to gain sufficient lead time to meet a post-1975 operation.

6.2 Recommendations for Future Effort

The architectural design presented in Chapter 5 is a solid base from which to proceed toward the goal of an operational system. A group of next steps suggests itself; for the most part, they represent continuing design applicable to the technology of computation in general, and would be valuable whether or not prototype development of the proposed configuration was planned.

As mentioned in the previous chapter, the architecture developed is well suited to both a family of multiprocessors of varied performance capability, and to a single-processor configuration. Whichever approach might be currently believed most promising, or indeed, if it was desired to postpone making a choice, the efforts outlined in the following sections would enable systematic progress to continue at modest cost.

6.2.1 Continuation of System Design

This study has provided a description of a basic computer organization, founded solidly on its technological predecessors. Certain areas have been identified as being of particular significance.

a) Maintenance of reasonable levels of internal bus traffic

through sizing of M1 and M2, and choice of an instruction set which takes full advantage of the hardware stack.

b) Provision of generalized, yet efficient, communication conventions for the I/O bus and devices.

c) Recognition of the failure recovery problem, and provision of a systematic hardware and software recovery design.

d) Provision of general, yet simple, operating system and file systems.

The following tasks are recommended.

6.2.1.1 Instruction Set

Design of the addressing structure, instruction formats, and instruction set are required. This task will additionally entail consideration of the register complement of the processor. Verification of compatibility with the chosen word length of 32 bits is inherently a part of the task.

6.2.1.2 Stack

Related to the preceding task, consideration of details of stack implementation in M1 will influence the instruction set design, and the treatment of data items of length other than one word. Stack interaction with M2 should be predicted on the basis of stack size and estimates of stack-switching frequency.

6.2.1.3 Buffer Memory

Details of information-handling in the buffer memory should be resolved. The influence of the instruction set, stack, and buffer size on the bus traffic should be formulated. Buffer sizing for constant bus traffic as a function of processor speed should be studied.

6.2.1.4 Internal Bus

Message formats, bus-access control, traffic levels, and interfaces should be established, and the adequacy of the bus-implementation technology proposed in Chapter 5 verified. Particular attention must be paid to reliability and error detection because of the central and solo nature of the bus.

6.2.1.5 I/O Controller

As the nature of the space station experiment packages and control requirements become more clearly defined, it will be possible to design the I/O controller in more detail. Message formats, interrupt structure, command multiplexing, and error handling are areas which require further design at the next stage of development. The definition of the standard interface element used by I/O devices falls into this category.

6.2.2 Software Design

Perhaps the clearest single problem area in the history of multiprocessor developments (and maybe computer systems generally) is the development of software which can operate the hardware at a satisfactory level of performance and reliability. It is Intermetrics' strong conviction that the most fundamental problems of this type can be avoided if the software development is undertaken with the same vigor and at the same time as the design of the hardware. If this is done, the two design efforts can interact freely, and stimulate changes when changes are least expensive. The following paragraphs contain specific recommendations.

6.2.2.1 Operating System Design

The philosophy proposed by Intermetrics for the space station operating system is presented in section 5.8. Implicit in this philosophy is the need to orient the operating system so that the overhead imposed is least when the frequency of use is highest. Protection must be air-tight, yet the operating system must be so direct that applications conceived years after the system becomes frozen have a very high probability of being compatible. The OS must allow those users who require it to benefit fully from the opportunities of extremely general and flexible sharing of procedures and data made possible by the multiprocessor type of organization.

6.2.2.2 Application Software Management

To assure compatability between the operating system and the management procedures which can be applied to the development of the vast quantity of semi-interdependent application software, it is essential that a software configuration management plan be prepared and employed for all software development for the space station. It should provide a common set of criteria and features for the software development, and should be enforced.

It is Intermetrics' belief that the best means for implementation of software conventions is to incorporate them

into the language translators used to prepare the executable code. This clearly requires that suitable languages be provided for the application software, and that their compilers be prepared so that they are capable of more than simple language translation.

Because of the length of the anticipated mission, software testing takes on even more importance than it conventionally has, since on-board testing should be provided to allow upgrading of virtually any part of the software without fear of system disruption.

6.3 Conclusion

We are aware that the centralized multiprocessor computer approach to the DMS application is not the only approach which has been recommended. Although it is clearly not within the scope of this study to analyze or perform a trade-off of alternate candidates, nor have we done so, it is our conclusion that a multiprocessor computer system can be designed, developed, and implemented to achieve the DMS functions in a cost-effective manner.

Even a cursory examination reveals several distinct advantages of this approach over one in which distributed computers are used:

a) Its cost should be lower, for a number of reasons. "Grosch's law" is an empirical observation that the cost of a computer is roughly proportional to the square root of its performance; thus one system with n units of power should prove less expensive than n systems each with one unit. Furthermore, the aggregate power of a distributed system must necessarily be greater than that of a central system, since each of the distributed units must be sized for its peak requirement, even if its average requirements is substantially lower. The peak load required of a central system is more closely related to the root-sum-square of the peak individual requirements than it is to their sum.

b) Its flexibility is greater, since it is inherently capable of expansion, and because it deals with a wide variety of applications via a simple, generalized interface.

c) Its reliability should be easier to achieve, since it is organized specifically for inherent failure-tolerance.

d) Its ability to implement communication among processes and users is vastly superior, since all are directly attached to the system and have access (when permitted) to common information in the common memories.

Other agencies are planning or using multiprocessing; the Navy AADC, the ASW aircraft S3A, the IBM 9020 for the FAA, to mention a few, plus the RCA-215, the Hughes 4400, CDC Alpha, and so on, which have been developed for general applications.

Intermetrics strongly recommends the adoption of the multiprocessor approach for the space station and space base Data Management Systems, and believes the design presented in Chapter 5 is an excellent candidate for these applications.

References for Chapter 6

1) Bobeck, A. H., et alia: "A New Approach to Memory and Logic - Cylindrical Domain Devices", Proceedings FJCC, Vol. 35, 1969, pp. 489-498.

2) Brown, J. R., Jr.: "First and Second Order Ferrite Memory Core Characteristics and Their Relationship to System Performance". IEEE Transactions on Electronic Computers, Vol. EC-15, No. 4, 1966, pp. 485-501.

3) Dailey, J. R. and Kuntzman, H. C., "The Impact of Technology and Organization on Future Computer Systems", Computer Design, February 1970, pp. 49-54.

4) Electronic News, January 26, 1970, pp. 1, 4-5.

5) Gold, H. S., and Pedersen, R. A.: "An Integrated Logic Gate with Subnanosecond Propagation Delay as a System Element", 1969 IEEE International Solid-State Circuits Conference, pp. 70-71.

6) Rudenberg, H. G.: "Large-Scale Integration: Promises versus Accomplishments - The dilemma of our Industry", Proceedings FJCC, Vol. 35 1969, pp. 359-367.

7) Wald, B., "Utilization of a Multiprocessor in Command and Control", Proc. IEEE, vol. 54, no. 12, December 1966.

8) Wegener, H. A. R., et alia: "An Integrated Nonvolatile Read-Write Memory with Addressing", Proceedings of 21st National Aerospace Electronics Conference, Dayton, Ohio, May 19-21, 1969, pp. 443-446.

9) Werner, G. E. and Whalen, R. M.: "A 375-Nanosecond Main Memory System Utilizing 7 mil cores". Proceedings FJCC, Vol. 26, 1965, pp. 985-993.

Appendix A

Survey of Paging and Segmentation Characteristics of Computer Systems

Many time-sharing systems have been implemented on computers which do not utilize paging. Among them are the DEC PDP-10, the Burroughs 5500 and 6500/7500, the Univac 1108, and the Control Data 6000 series. Other manufacturers have elected to implement paging; a number of these are reviewed below.

A.1 The Control Data 3300

The memory in the CDC 3300 is logically divided into pages of 2048 48-bit words. A special fast core memory is used to contain the physical addresses of pages in storage, indexed by the upper bits of the logical address from the program. Pages are further divided into quarters; storage may be allocated on the basis of this smaller quantum. Two additional bits are provided in each entry of the fast-core page table for quarter-page addressing. These two bits are added to two bits of the program-specified address with wrap-around, so that a three quarter page requirement can be satisfied by a page with, say, only quarters 3, 4 and 1 available.

A.2 The Control Data 3800

The 3800 paging mechanism uses an allocation memory (AM) which contains 128 13-bit words. Three of the bits from each entry are used for page access-protection, and to provide write protection for the lower and upper halves of the page. Seven to ten of the remaining bits are used for relocation, depending on the page size. A four-position switch is used to set the page size to 256, 512, 1024, or 2048 words, and to control the selection of address bits used to index into the AM. In all cases, seven bits are used to specify the location in the AM, but the position in the logical address from which they are taken is varied by the switch setting.

A.3 The XDS Sigma 7

Like the IBM 360, the Sigma 7 uses 8-bit bytes, 4-byte words, etc. However, the addressing scheme is quite different. The virtual memory addressed by the program is limited to 512K bytes, which is 256 2048-byte pages. A high-speed 256-byte memory map is provided to perform address translation; eight bits of the virtual address select one of these bytes, whose contents are substituted for the original eight bits of the

virtual address to form the physical address.

Associated with each page is a two-bit access control code, which is used to selectively inhibit non-executive programs from reading, executing, or writing page contents. A 2-bit lock and key protection feature is also provided; a program can write into a given block if lock and key values match (the lock is associated with storage, the key with the process being executed), or if either is zero.

A.4 The RCA Spectra 70/46

The Spectra 70/46 is basically a Spectra 70/45 with memory address translation hardware added. The virtual memory and paging facilities are achieved using a translation memory. Control bits in each translation memory entry indicate whether the corresponding page has been written into or accessed. These bits are set automatically by hardware. Each translation memory entry also includes a "usable" bit, indicating whether the page is in memory, and the physical address of that page in memory.

The translation memory has 512 entries, one for each 4K-byte page; the limit of virtual memory is thus two million bytes. Although the virtual address is broken down into what are referred to as segment and page fields, addressing is one-dimensional. The segment concept refers only to the fact that eighteen-bit address arithmetic is used. Specifically, nine bits from the address are used to select entries from the 512-halfword translation memory. Each entry, in turn, supplies six bits which are combined with the 12-bit displacement in the original address. Thus, no more than 64 pages (one "segment") can be contiguously addressed.

By convention, the first four segments, or 256 pages of virtual memory, are available as users' virtual memory. The other four segments are not available to users' programs, but are used by the control program. This system virtual memory is always allocated for system and shared code. Its mapping is resident in the translation memory and need not be modified as control is passed from one task to another. When a task is to be given control of a processor, the necessary portion of the first half of the TM entries (the size of its assigned virtual memory) is loaded.

A.5 The Burroughs 5000/5500

The Burroughs 5000 was one of the first computers to use the segment concept. The segments are variable in length, but have a maximum size of 1024 words of 48 bits. Users of the B5500 are not supplied with an assembler; thus, all programs are expected to be written in compiler languages. The system

programming was done in a language called Extended ALGOL. Programs are segmented by compilers at the level of ALGOL blocks or COBOL paragraphs. Arrays are also compiled as separate segments. The segment is used as the unit of memory allocation. Not all segments have to be present in the core memory for the program to begin running. When reference is first made to a segment, the segment is fetched by the executive in response to an interrupt.

Each program is assigned a program reference table (PRT), pointed to by a special register in the CPU. Each segment of the program is represented by a PRT entry, which contains the base address, the length of the segment, the starting location relative to the base, and an indication of whether the segment is currently present in memory. The entries in the PRT are called descriptors by Burroughs. Core selection strategy to prevent fragmentation includes choosing the smallest available block of sufficient size. Because of the way in which segments are formed, the average segment size is on the order of one or two hundred words.

A.6 The GE 645, and Multics

The Multics project at MIT was the innovator of two-dimensional addressing with paging (cf. Chapter 2). The basic motivation behind the combination was the desire to permit information sharing in a more automatic and general matter. Consider, for example, the problems involved in a non-segmented system when a file is to be shared: typically a copy of the desired information is provided to each user in response to I/O requests he issues. Any modification or updating is done on each copy, and is reflected in the original file only upon completion of further I/O requests. Thus, logically acceptable updates performed by different users at nearly the same time can prove disastrous. In Multics, on the other hand, each file is a segment. When a file is initially referred to by a user, it becomes "active". Initial references by subsequent users will find the segment active; only one page table will ever exist for the file. If a user refers to a given address in the file, the operating system automatically finds and fetches the page into memory. Thus, the notion of copy is irrelevant; a file page either is present in memory, or on secondary storage, at the pleasure of the core-management routines.

The implementation of this system is quite complex. An address can be considered to comprise two major parts: segment number, and the word offset within the segment. In operation, each of these parts is further divided into page and page-offset parts. A key processor register, the "descriptor-segment base register" (dbr), contains a value unique to each process: the address of the page-table for the "descriptor segment". This segment contains a list of pointers to page-tables of segments

known to that process. In address decoding, the page number part of the segment number is added to the pointer in the dbr to select a word in the descriptor segment page table. This word contains the address of the page of the descriptor segment; the low-order part of the segment number is added to this address to locate the entry which points to the page table for the appropriate segment. This page table is unique to the segment regardless of the number of users to which the segment is currently known.

In a similar manner, the upper half of the segment offset is used to select an entry from the segment page table, which points to the core location of the page; the lower half of the segment offset is then used to finally address the word wanted.

Prior to each lookup in a page-table, the hardware checks the index to be used against a length-limit contained in the pointer to the page-table; if the index is invalid, the operation is trapped, and control is passed to the supervisor for process termination. Each page-table entry contains a bit used to alert the supervisor if the page is not physically in core, by means of a page-fault. The supervisor responds by fetching the page, and then returning control to the user. If a selected segment descriptor word similarly indicates the absence of the segment page table, this triggers a missing-segment fault, which requires the supervisor to make the segment "known".

The GE-645 provides the means to use 1024 or 64 word pages, and to use unpaged segments. Although the original Multics implementation used both page sizes, the use of 64-word pages was abandoned to enhance system performance. Unpaged segments are used only in the part of the supervisor which is core-resident. Sixteen associative registers are provided in the processor to retain recently-used descriptor-segment and page table entries, to speed subsequent references.

A.7 IBM System 360/Model 67

This system, which is generally compatible with the rest of the 360 line, has additional features to enhance its time-sharing utilization. Memory is divided into 4096-byte pages (1024 32-bit words). The IBM operating system TSS/360 takes up about 90 pages, so that a 512K machine has only about 40 pages left for user-multiplexing. As a result, performance of systems with larger memory has been substantially better.

The addressing in the 67 is two dimensional; upper bits of the address select a word in a segment table which points to the page table for that segment. The remainder of the address specifies a word in the segment by indicating a page table entry and an offset within the page. Although in the 360 line addresses are 24 bits long, a 32-bit mode is optionally available on the 67 to increase the number of addressable segments from 16 to 4096.

Appendix B

Physical Characteristics

The physical characteristics of a computer are strongly influenced by the requirements imposed upon it by the operational environment. Size and weight limitations are chronic problems in space applications; they become the main forcing functions in determining the ultimate physical characteristics. To these may be added temperature, pressure, humidity, shock, vibration, "g", and so on. The logistical requirements of modularity, standardization, maintenance, repair, etc., add further constraints. Accommodating these constraints without compromising the required performance is a burden that falls upon the technologies of implementation and manufacture.

It is obviously impossible at this time to formulate a realistic projection of the physical properties of the computer proposed in this report without realistic estimates of its expected environment, or of the performance requirements. This study has concentrated on an evaluation of basic configurations. The design presented in Chapter 5 has been chosen as the optimum compromise of the conflicting factors discussed in other chapters. The choice was made without specific assumptions about the implementation technology. However, it is the application of this technology that will contribute substantially to the physical characteristics of the computer. Over the next 5 to 10 years this technology will experience periods of very rapid development, so-called "breakthroughs", which will diminish the accuracy of predictions based on current rates of development.

Nevertheless, the following heuristic approach to estimating the proposed computer size and weight is presented to give at least some idea of its scale. The weight, volume, power dissipation and performance of a number of current aerospace computers have been normalized to allow them to be compared and plotted against time. The machines chosen are listed in Table B.1. Although some of them have more advanced features (e.g., floating point arithmetic) than others, they are comparably organized, and the memory cycle time is taken as an indication of processing speed. The basic memory size is used in determining the performance factor, even though the capability of many of these computers can be extended considerably by adding more storage.

The performance is expressed by the factor K, where

$$K = \frac{\text{Memory size (bits)}/10^6}{\text{Memory cycle time } (\mu s)}$$

I.e., a 10^6 bit machine with a 1 μs MCT has one unit of performance capability.

Normalized weight, volume and power are calculated from:

$$M^* = M/K$$

$$V^* = V/K$$

$$W^* = W/K$$

and are plotted in Figures B.1 through B.3. The straight line plots are intended to suggest trends and are not mathematically derived.

The validity of an extrapolation to 1975, which must be the cut-off point for designs that are to be operational 2-3 years later, is questionable because:

a) It is unlikely that the rate of development of the current technologies common to all the plotted computers will be maintained at a uniform rate for the next 5 years.

b) It is very likely (as mentioned above) that in a five year period, quite novel techniques of implementation will be developed, to which these graphs may have no relation.

Nevertheless, if these factors are ignored, and if it is assumed that K is linear for all values of memory size and processing speed, the subject computer will, by 1975 standards, possess the following characteristics, per processor (assuming $M2 = 10^7$ bits per processor, cycle time of 100 ns, and an efficiency factor of 70% - see 5.2.1):

$$K = \frac{10^7/10^6}{10^{-1}} \times 0.7 = 70$$

Weight = 700 lbs.

Volume = 7 cu. ft.

Power = 700 watts

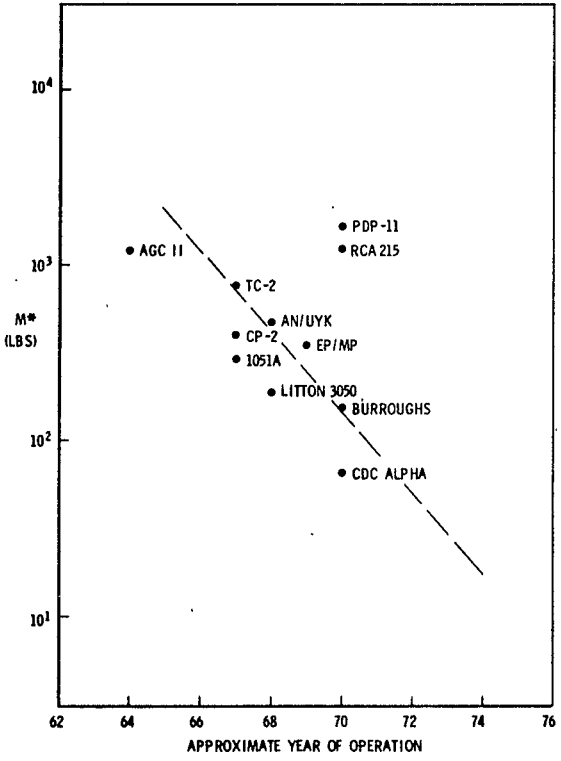


Figure B.1
Normalized weight vs. operational date

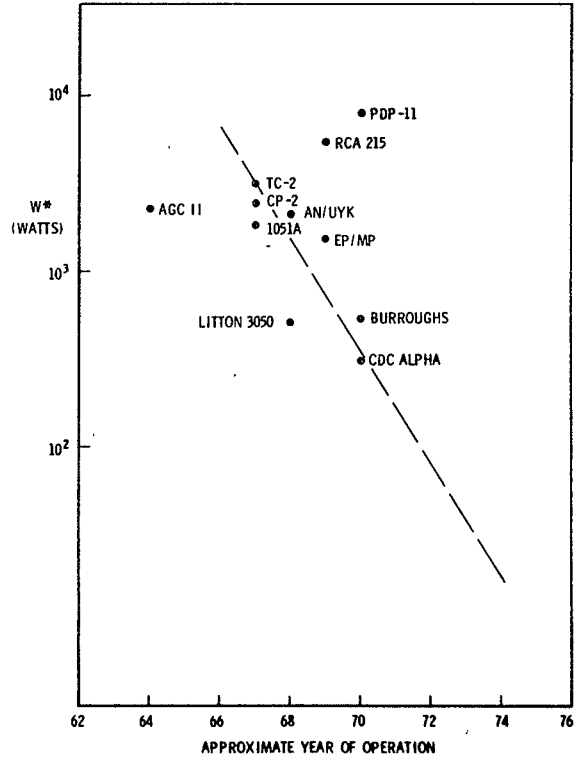


Figure B.2
Normalized Power vs. operational date

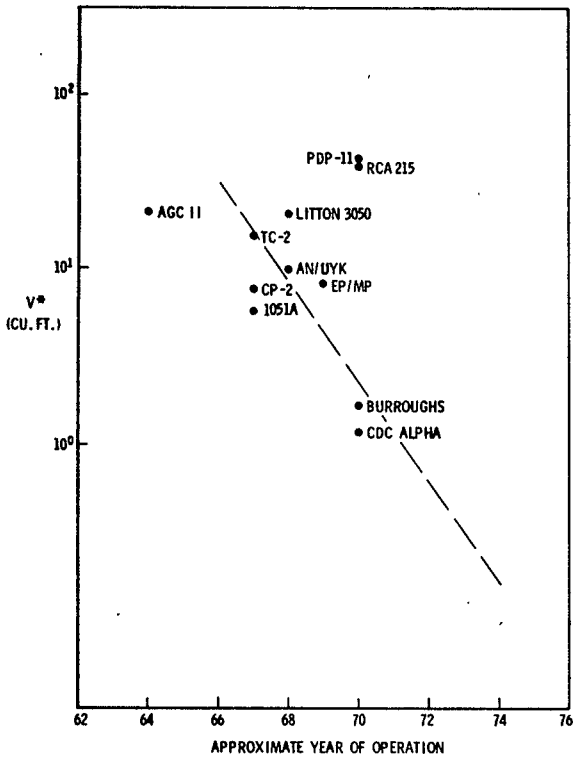


Figure B.3
Normalized volume vs. operational date

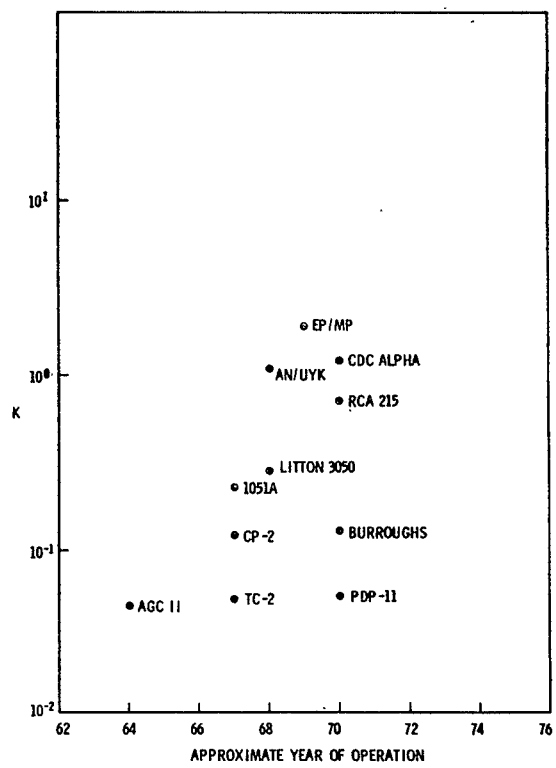


Figure B.4
Normalized performance vs. operational da

Table B.1

Computer	Approximate Year Of Operation
IBM EP/MP	69
Nortronics 1051A	67
IBM TC-2	67
RCA-215	70
Univac AN/VYK	68
Burroughs	70
CDC ALPHA	70
AGC II	64
Litton 3050	68
IBM CP-2	67
DEC PDP-11	70

Appendix C

Additional Bibliography

Listed in this appendix are a number of papers studied in the course of the contract which were found to have relevant content. The list does not include papers cited in the chapter reference lists.

1) Ashenhurst, R.L., "The Maniac III Arithmetic System", Proc. SJCC, 1962, Vol. 21. National Press, Palo Alto, California.

2) Barron, D.W., Recursive Techniques in Programming. American Elsevier. 1968.

3) Batson, A., Ju, S., and Wood, D.C., "Measurement of Segment Size", ACM Second Symposium on Operating System Principles. Princeton, N.J. Oct. 20-22, 1969. pp. 25-29.

4) Beelitz, H.R., Levy, S.Y., Linhardt, R.J., Miller, H.S., "System Architecture for Large-Scale Integration", Proc. FJCC, 1967, Vol. 31. Thompson Books, Washington, D.C.

5) Belady, L.A., and Kuehner, C.J., "Dynamic Space-Sharing in Computer Systems". CACM 12, 5 May 1969, pp. 282-288.

6) Bensoussan, A., Clingen, C.T., and Daley, R.C., "The Multics Virtual Memory", Second Symposium on Operating System Principles. Princeton, N.Y. Oct. 20-22, 1969.

7. Burroughs Corporation, B8500 System Reference Manual, BJ-8, May 1967.

8. Control Data Corporation, 3800 Computer System Reference Manual. Oct. 1965.

9. Corbato, F.J., and Saltzer, J.H., "Some Considerations of Supervisor Program Design for Multiplexed Computer Systems", MIT Memo MAC-M-372. May 1968.

10. Daley, R.C. and Dennis, J.B., "Virtual Memory, Processes, and Sharing in Multics", CACM 11,5, May 1968.

11. Dennis, J.B., "Segmentation and the Design of Multiprogrammed Computer Systems". J. ACM 12, 4. 1965. pp. 589-602.

12. Dennis, J.B., and Van Horn, E.C., "Programming Semantics for Multiprogrammed Computations", CACM 9,3, March 1966.

13. Dijkstra, E.W., "The Structure of the 'THE' - Multiprogramming System", CACM 11,5, May 1968, pp. 341-346.

14. Fine, G.H., Jackson, C.W., and McIsaac, P.V., "Dynamic Program Behavior under Paging" Proc. ACM 21st National Conference, 1966, pp. 223-228.

15. Fuchel, K., and Heller, S., "Considerations in the Design of a Multiple Computer System with Extended Core Storage", CACM 11,5, May 1968, pp. 334-340.

16. Glaser, E.L., Couleur, J.F., and Oliver, G.A., "System Design of a Computer for Time Sharing Applications", Proc. FJCC. 1965, Vol. 27, pp. 197-202.

17. Graham, R.M., "Protection in an Information Processing Utility", CACM 11,5, May 1968.

18. Hauch, E.A., and Deut, B.A., "Burroughs' B6500/B7500 Stack Mechanism", Proc. SJCC 1968, Vol. 32, pp. 245-251.

19. IBM System/360 Model 67 Functional Characteristics, Form A27-2719-0. 1967.

20. Kerner, H., Gellman, L., "Memory Reduction Through Higher Level Language Hardware", AIAA Aerospace Computer Systems Conference, paper 69-963, Los Angeles, Calif., Sept. 1969.

21. Kopf, J.O., and Plauger, P.J., "JANUS: A Flexible Approach to Realtime Time-Sharing", Proc. FJCC, 1968, Vol. 33, pp. 1033-1042.

22. Kuehner, C.J., and Randell, B., "Demand Paging in Perspective", Proc. FJCC, 1968, Vol. 33, pp. 1011, 1017.

23. Lampson, B.W., "A Scheduling Philosophy for Multiprocessing Systems", CACM 11,5, May 1968, pp. 347-360.

24. Lampson, B.W., "Dynamic Protection Structures", Proc. FJCC, 1969, Vol. 35, pp. 27-38.

25. Linde, R.R., Weissman, C., and Fox, C.E., "The ADEPT-50 Time-Sharing System", Proc. FJCC, 1969, Vol. 35, pp. 39-50.

26. Mullery, A.P., and Driscoll, C.C., "A Processor Allocation Method for Time-Sharing", CACM 13,1, Jan. 1970, pp. 10-14.

27. Oppenheimer, G., and Weizer, N., "Resource Management for a Medium Scale Time-Sharing Operating System", CACM 11,5, May 1968.

28. Randell, B., and Kuehner, C.J., "Dynamic Storage Allocation Systems", CACM 11,5, May 1968, pp. 297-306.

29. Randell, B., "A Note on Storage Fragmentation and Program Segmentation", CACM 12,7. July, 1969. pp. 365-372.

30. Rappaport, R.L., "Implementing Multi-process Primitives in a Multiplexed Computer System", M.S. Thesis, MIT, 1968, MAC-TR-55.

31. Saltzer, J.H., "Traffic Control in a Multiplexed Computer System", Sc. D. Thesis, MIT, 1966, MAC-TR-30.

32. Sayre, D., "Is Automatic Folding of Programs Efficient Enough to Displace Manual?", CACM 12, 12. Dec. 1968. pp. 656-660.

33. Vareha, A.L., Rutledge, R.M., and Gold, M.M., "Strategies for Structuring Two Level Memories in a Paging Environment". ACM Second Symposium on Operating System Principles. Princeton, N.J. Oct. 20-22, 1969. pp. 54-59.

34. Vyssotsky, V.A., Corbato, F.J., and Graham, R.M., "Structure of the Multics Supervisor," Proc. FJCC, 1965, Vol. 27, pp. 203-212.

35. Wald, B., "Utilization of a Multiprocessor in Command and Control", Proc. of IEEE 54, 12. Dec. 1966, pp. 1885-1888.

36. Wallace, V.L., and Mason, D.L., "Degree of Multiprogramming in Page-on-Demand Systems", CACM 12, 6., June 1969, pp. 305-318.

37. Wilkes, M.V., "The Growth of Interest in Micro-programming- A Literature Survey", Comp Surveys 1,3. Sept. 1969. pp. 139-145.

38. Wirth, N., "On Multiprogramming, Machine Coding, and Computer Organization", CACM, 12,9, Sept. 1969, pp. 489-498.